Manipulator Control Using Real Time Extensions for Windows NT

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Masters of Engineering at The University of Canterbury.

Stuart D. Donald

August 2001

Abstract

A large hydraulic Unimate 2000B robot was donated to the University of Canterbury by Industrial Research Ltd. Christchurch. The original point-to-point control system was spread across 8 A3 size large circuit boards. A desktop personal computer running Microsoft Windows NT on a VenturCom RTX kernel is used with a FPGA and custom A5 sized circuit board to replace the original system with path control. VenturCom's Real Time extensions to the NT operating produced satisfactory Real Time operation for controlling the hydraulic manipulator, with a worst case interrupt latency of 27μs on a 233MHz Pentium II, and interrupt rates of up to 6kHz.

Full path control via a graphical user interface has been developed to provide the Robot with greater flexibility than was available with the original point-to-point control system. A PI controller was used for the control algorithm, with the tuning constants found by using frequency response analysis. This type of controller was found to work to a reasonable level of accuracy, although further investigation into an adaptive model controller could perhaps provide better performance.

Nomenclature

DSP	Digital Signal Processor
FIFO	First In First Out Buffer
FPGA	Field Programmable Gate Array
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
IPC	Inter-Process Communication
ISR	Interrupt Service Routine
MIMO	Multi Input Multi Output
NT4	Windows NT 4
PC	Personal Computer
PD	Proportional Derivative Controller
PI	Proportional Integral Controller
PWM	Pulse Width Modulation
RTL	Real Time Linux
RTOS	Real Time Operating System
RTSS	Real Time Sub System
RTX	Real Time eXtensions
SISO	Single Input Single Output

Contents

Abstract ..... **I**

Nomenclature ..... **II**

1 Introduction **1**

1.1 Brief Description of Project 1
1.2 Real Time Control 1
1.3 Software 2
1.3.1 Brief Description of Software Developed 2
1.4 Hardware 3
1.4.1 FPGA 3
1.4.2 Custom Circuit Board 3

2 Real Time Extensions for Windows NT **5**

2.1 Performance 7
2.2 IPC 7

3 Unimate 2000B **9**

3.1 Manipulator Hydraulics 9
3.2 Manipulator Safety Measures 10
3.3 The Circuit Board 10
3.4 The Absolute Grayscale Encoders 11
3.5 Kinematics of Unimate Model 2000B 12
3.6 Inverse Kinematics 14
3.6.1 Multiple Solutions to Inverse Kinematics 15
3.7 The Jacobian 16

4 Controller Design **19**

4.1 System Model 19
4.2 PI Control 22
4.3 Frequency Response of System 23
4.4 Controller Design 28
4.5 Performance of Controller 28
4.5.1 Uncoupled Ramp Response 28
4.5.2 Coupled Ramp Response 32

5 Software Design **35**

5.1 Real Time Software 35
5.1.1 Cache Memory 35
5.1.2 How Cache Works 36
5.1.3 Design of code to Maximise Cache Hits 37
5.1.4 Cache and Problems with Real Time applications 38
5.2 Communication between RTSS & win32 processes 39
5.3 Real Time Software 40
5.4 Graphical User Interfaces 41
5.4.1 Main Control GUI 42
5.4.2 Path Generation GUI 47

6 Trajectory Generation **50**

6.1 Outline 50

6.1.1 Step 1 Define End Points:....53
6.1.2 Step 2 Generate Parametric Path Through End Points:....53
6.1.3 Step 3 Convert Parametric Path to Time Dependant Path:....55
6.1.4 Step 4: Create Joint Space Trajectory....58
6.1.5 Sample Trajectory....61
6.2 Check Maximum Joint Parameters....63
6.2.1 Joint Position....63
6.2.2 Joint Velocity....64
6.2.3 Joint Acceleration....64
6.3 Path Deviation....65
6.3.1 Analytical Solution of Path deviation:....65
6.3.2 Approximate measure of Path Quality....67
6.4 Create Virtual Robot Workspace....68
7 Discussion....70
7.1 Hardware difficulties....70
7.2 Controller....70
7.3 Integration with Matlab....72
7.4 Educational Use of Robot....72
7.5 Comparison with DSP....73
8 Conclusion....74
9 References....76
A. Inverse Kinematics....77
B. Robot Motions....82
C. Derivation of Jacobian....83
D. Spline Interpolation....86
E. Thomas Algorithm....88
F. Fix for Unreliable Encoder....89
G. Software Start-up Instructions....91

List of Figures

Figure 2-1Histogram of Interrupt Latency 7

Figure 3-1The Unimate 2000B hydraulic robot..... 9

Figure 4-1 Up/Down Axis Side Elevation.....19

Figure 4-2 Rotate Left/Right Plan View.....19

Figure 4-3 In/Out Side Elevation 20

Figure 4-4 Plant Model.....21

Figure 4-5 Simplified Model of Plant 22

Figure 4-6 PI Controller (Frequency Domain) 22

Figure 4-7 PI Controller (Digital Implementation).....23

Figure 4-8 Bode Plot - Rotate (Boom In).....24

Figure 4-9 Bode Plot - Rotate (Boom Out) 24

Figure 4-10 Bode Plot - Up/Down (Boom In).....25

Figure 4-11 Bode plot - Up/Down (Boom Out).....25

Figure 4-12 Bode Plot - In/Out 26

Figure 4-13 Bode Plot - Bend 26

Figure 4-14 Bode Plot - Yaw.....27

Figure 4-15 Bode Plot - Swivel.....27

Figure 4-16 Rotate Ramp Response.....29

Figure 4-17 Up/Down Ramp Response.....29

Figure 4-18 In/Out Ramp Response.....30

Figure 4-19 Bend Ramp Response.....30

Figure 4-20 Yaw Ramp Response.....31

Figure 4-21 Swivel Ramp Response 31

Figure 4-22 Rotate Coupled Ramp Response.....32

Figure 4-23 Up/Down Coupled Ramp Response.....33

Figure 4-24 In/Out Coupled Ramp Response.....33

Figure 4-25 Bend Coupled Ramp Response....34
Figure 4-26 Yaw Coupled Ramp Response....34
Figure 5-1 Model of Typical Cache Set-Up....36
Figure 5-2 Inter-process Communication....40
Figure 5-3 Layout of Real-Time Software Application....41
Figure 5-4 Main Robot Control GUI....42
Figure 5-5 PWM Mode Property Page....43
Figure 5-6 Frequency Mode Property Page....43
Figure 5-7 Step Mode Property Page....44
Figure 5-8 Ramp Mode Property Page....44
Figure 5-9 Track Sin Property Page....45
Figure 5-10 Path Generation GUI....48
Figure 5-11 Path Generator Dialog Control Event Map....49
Figure 6-1 Overview of Path Generation....52
Figure 6-2 Time Law for Parametric Path....58
Figure 6-3 End Effector Trajectory....61
Figure 6-4 Joint Trajectory....62
Figure 6-5 Geometric Interpretation of Path Deviation Measure....67
Figure 6-6 SolidWorks Model of Unimate 2000B....69
Figure 7-1 Adaptive Controller....71
Figure A-1 Joint Origons....77
Figure B-1 Robot Motions....82

List of Tables

Table 1 Denavit Hartenburg Constants for the Unimate 2000B.....14

Table 2 Parameters for PI Controller with a system sample time of 1ms 28

1 Introduction

1.1 Brief Description of Project

The objective of this project is to assess the use of Real Time Extensions for Windows NT4 (RTX) specifically for the use in robotics. This investigation uses the specific example of retrofitting path control to a six-axis hydraulic manipulator. RTX for windows NT4 allows real time processes to be run in a windows environment, thus real time control of the robot can be achieved on a general purpose PC, with the normal windows interface used to communicate robot performance to the user.

The operation of RTX was initially assessed to provide information on stability and performance. A graphical user interface and a real time controller was then designed for the robot. This software implements path-planning capabilities, and provides superior control of the robot as well as a user-friendly interface.

1.2 Real Time Control

There are several definitions of real-time, most of them contradictory. Unfortunately the topic is controversial, and there doesn't seem to be 100% agreement over the terminology. The following definition seems most appropriate for the purposes of this project:

"A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred." [Gillies]

Windows NT on its own can offer soft real time, Windows NT with RTX offers hard real time. A hard real time system means the type of real-time system discussed above, soft real-time means systems which have reduced constraints on "lateness" but must still operate very quickly and repeatably. A good example is a robot that has to pick up something from a conveyor belt. The piece is moving, and the robot has a

small time window in which to pick up the object. If the robot is late, the piece won't be there, and thus the job will have been done incorrectly even though the robot went to the right place. If the robot is early, the piece won't be there, and the robot may block it. The system would be defined as hard real time if the robot arriving late causes completely incorrect operation. It would be soft real time if the robot arriving late meant a loss of throughput. Much of what is done in real time programming is actually soft real time system. Good system design often implies a level of safe/correct behaviour even if the computer system never completes the computation. So if the computer is only a little late, the system effects may be somewhat mitigated.

1.3 Software

Microsoft Windows NT 4.0 and VenturCom's RTX 4.3 are used in this implementation. The advantage of using a PC to control the robot is that custom control and user interface software may be easily created. All of software has been created in C++ using Microsoft's Visual C++ environment. The real time code is restricted to being written in C/C++ however the user interface could have been created in a number of different languages, including Java, Visual Basic, or Matlab (although the use of these languages would still require the use of win32 dynamic link libraries written in C++ to allow communication with the real time process). The RTX API does not include support for any GUI related calls. Hence the GUI is created as a win32 process and communicates with the RTSS (Real Time Sub System) process through shared memory.

1.3.1 Brief Description of Software Developed

The robot GUI allows control of the robot in a number of different modes, including direct PWM manipulation and path tracking. Inter-Process Communication between the GUI win32 process and the RTSS process is achieved through shared memory. Offline path generation is created using another C++ application. A path is created from a series of end effector matrices. The matrices may be either defined relative to the base frame or the previous frame. The end effector matrices may be entered directly in the form of Euler or RPY angles and tip position. The end effector matrix can also be measured from the robots current position if required. A cubic spline is then fitted to the end effector path. The inverse kinematics are then solved and a joint

space spline is fitted through each of the resulting points. The Real Time ISR uses a PI algorithm to calculate the PWM value required to set the actuator current for each of the solenoids. The control effort for each solenoid is recalculated at a rate of 1kHz, however it should be noted that only one encoder can be read at a time, hence the ISR is called at a rate of 6kHz, which allows a preselected encoder 166μs to settle before being read.

1.4 Hardware

The PC used for this project is a 233MHz Pentium II MMX, with 128 Mb RAM. The other hardware used in this application consists of a Xilinx field programmable gate array (FPGA), and a custom A5 sized circuit board.

1.4.1 FPGA

The FPGA has been interfaced to the ISA bus in the PC so that the win32 and RTSS processes can interact directly with the FPGA. The FPGA board fulfils two main functions:

- generating the interrupt timer
- generating the PWM signals
- interface for reading and selecting the 14 bit encoders

The FPGA board has a 12MHz crystal clock, which is used to generate the interrupt timer. The FPGA board sets a register flag at regular intervals as specified by the program at run time. The RTSS process interrupt is connected to this register, and is triggered on the activation of this flag which also resets the watchdog timer. At the conclusion of the ISR the register flag is reset, and the Real Time process is suspended until the flag is again set.
The FPGA also generates the PWM signals required to drive the solenoid valves. The FPGA takes the required PWM Duty cycle (0 - 100%) and direction from the software, and converts it to a digital signal which is sent to the external circuit board.

1.4.2 Custom Circuit Board

The external custom circuit board provides optical circuit isolation as well as amplification of the PWM signals generated by the FPGA board. The Board also

houses a relay that shuts down the robot if the robot workspace is compromised, or the computer crashes, i.e. the FPGA register flag watch dog timer is not reset at the right time.

2 Real Time Extensions for Windows NT

Real Time eXtensions (RTX) for Windows NT4.0 (WinNT) addresses the real-time development and execution of time critical applications that require high performance and real-time determinism. RTX allows WinNT to function as both a general-purpose operating system and a high-performance real-time operating system, at the same time, on the same computer.

WinNT is designed as a business operating system, and while developers of real-time applications desire the benefits that come from using Windows, they remain focused on real time development and execution. VenturCom's RTX technology enables WinNT to address these needs.

RTX Features

- Supports RTWinAPI - the common Real-time Windows CE, Windows NT, and Windows 2000 API (application programming interface).
- Real-time application wizard and source code to numerous sample programs.
- Tools for measuring worst-case response times for performance evaluation of various hardware platforms.
- Execution on uni-processor and multiprocessor Windows NT and Windows 2000.
- Continuation of real-time processing even after Windows NT/Windows 2000 exception handling (blue screen).
- API enhancements for direct access to hardware.
- Develop and debug applications as standard win32 applications or real-time processes with standard Microsoft tools.
- Applications can start seconds into the boot cycle for quick recovery of control tasks.

RTX API's provide a complete and powerful win32-based programming interface designed to support all real-time applications and real-time device drivers. The RTX API consists of both win32 functions specially selected to provide real-time

performance, and other functions required for direct interfacing to hardware devices. Standard development tools, such as Microsoft Visual C/C++, are used to develop RTX applications. The C and C++ languages are supported by RTX along with the C runtime library, various C++ libraries, and structured exception handling.

The RTX API includes the following functional areas:

- Process and Thread Management
- Fixed Priority Scheduling
- Inter-Process Communication and Synchronization
- Memory Management
- High Speed Clocks and Timers
- Port I/O
- Physical Memory Mapping
- Interrupt Management

A Real-time SubSystem (RTSS) provides the core functions and resource management for deterministic control of Windows NT under RTX. The RTSS enables deterministic performance by utilizing the timer and interrupt management services of the HAL[1]. This provides the RTSS thread scheduler with the fastest possible response to device interrupts and enables the lowest possible latency scheduling of RTSS threads. The RTSS environment is able to pre-empt Windows NT at any time. Finally, RTSS processes can continue executing after a Windows NT stop event, until an orderly shutdown, (the sequencing of critical hardware), is completed.

[1] Windows NT is designed to run with a variety of CPUs and hardware platforms. To solve this portability issue, Windows NT isolates the kernel code from the hardware by using an isolation layer referred to as the hardware abstraction layer, or HAL. The HAL presents a "virtual machine" to the kernel, executive, and device drivers. It maps this virtual machine onto the underlying hardware in an efficient manner and eliminates the need for Windows NT to provide hardware specific kernels, which improves overall efficiency.

2.1 Performance

The performance of a RTOS (Real Time Operating System) can be assessed by measuring the interrupt latencies. The interrupt latency is a measure of the time taken from when an interrupt signal is generated to when the ISR (Interrupt service routine) begins to be executed. The interrupt latency has been measured using the timer on the FPGA board. When an interrupt is triggered a counter on the FPGA board begins, at the beginning of the ISR the counter value is read and written to shared memory for access by a WinNT application. An average interrupt latency of 14 μs was recorded using a 233MHz Pentium II. The maximum and minimum were 26 μs and 12 μs respectively. Even though the average interrupt latency is 14 μs the software needs to be designed for the worst-case scenario, hence the 26 μs latency needs to be considered for design purposes.

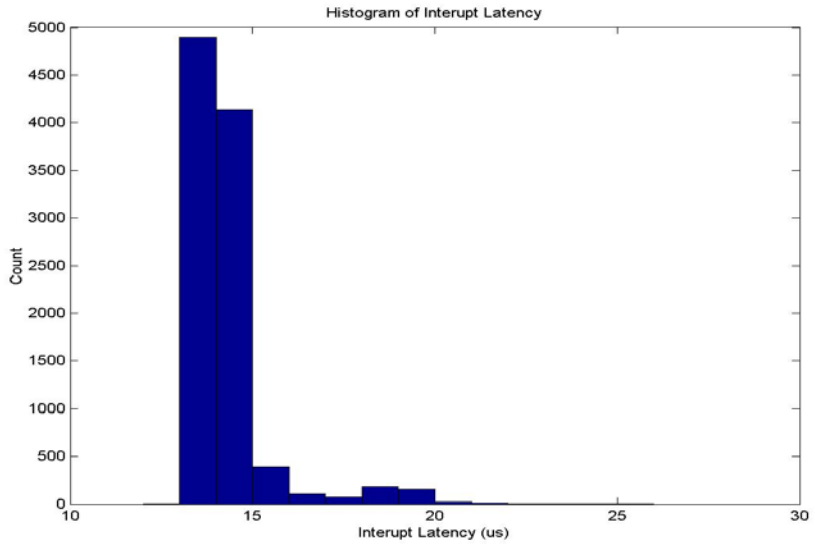

Figure 2-1Histogram of Interrupt Latency

2.2 IPC

RTSS and win32 Processes communicate and share duties by utilizing the RTX API between processes. Applications can be created that partition their time-critical functions to run in the RTSS environment and non-time-critical functions to run in the

WinNT environment. Processes in both environments can communicate and synchronize through the shared RTX inter-process communication (IPC) objects. Processes in the RTSS environment perform the high-speed deterministic control functions of the application. The processes in the WinNT environment utilize the complete range of win32 functions such as graphical displays, network communications, and data storage. This is the software model that is utilised throughout this research.

3 Unimate 2000B

3.1 Manipulator Hydraulics

All six axes of the Unimate 2000B manipulator are powered by hydraulic actuators, which are under control of servo valves. Hydraulic actuators for In-Out and Down-Up motions are connected directly to their respective loads. For rotation, a rack and pinion converts linear travel of the hydraulic rams to rotary motion. The motion of Bend, Yaw and Swivel are transmitted by systems of chains, gears and shafts since the wrist is moved by the arm and the wrist has to be kept at a minimum weight. A ball nut and spline shaft arrangement was used to transmit motion to the wrist axes.

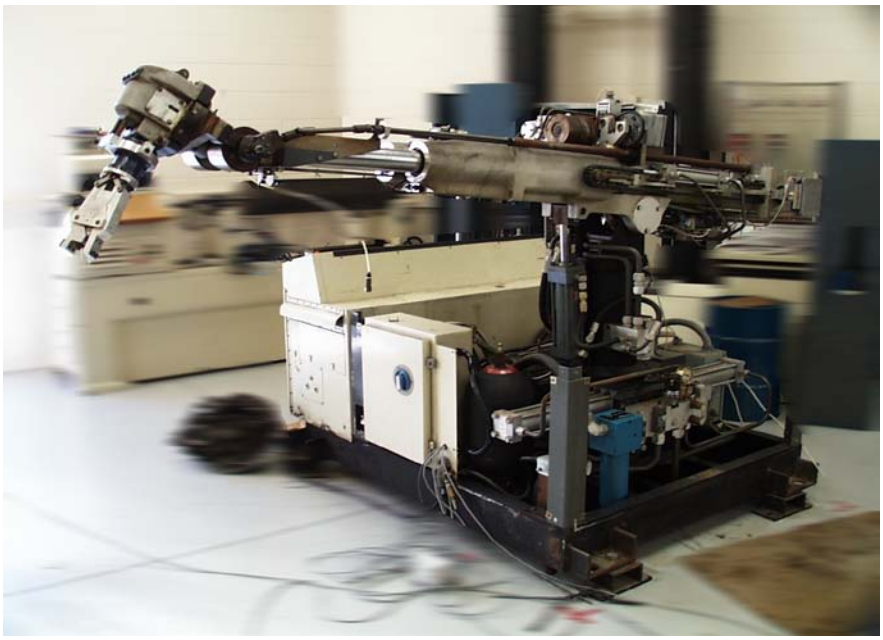

Figure 3-1The Unimate 2000B hydraulic robot.

The hydraulic power is generated using a type of vane pump that is powered by a 10 hp (7.5 kW) electric motor. The accumulator has been charged with dry nitrogen to a pressure of 525 psi (3.5 Mpa). This extra pressure ensures that the system pressure will be maintained when instantaneous flow demand exceeds the pump output flow capacity. Each of the 6 Moog servo valves (four way infinite position valves), is controlled by an electrical signal. The polarity of the signal controls the direction of the movement, while the magnitude of the signal controls the size of the valve

opening. A larger valve opening means a higher flow rate, therefore a faster joint movement. A servo valve directs the hydraulic fluid to one side of a hydraulic actuator and opens the opposite side for the return flow. When moving in the opposite direction, the connections are reversed.

The Rotary and In-Out actuators are slightly different. The rotary actuator consists of two actuators at opposite ends of the rack. Hydraulic fluid is admitted from the piston side of each actuator just as though it was a single piston in a single actuator. The In-Out actuator has fluid at system pressure at the rod side of the piston at all times. The servo valve controls fluid flow to or from the piston side only of the Out-In actuator. The Swivel motion utilises a hydraulic motor, not a linear actuator.

3.2 Manipulator Safety Measures

A physical barrier in the form of a chain has been installed across half of the hydraulics lab to prevent people from going into the manipulator workspace. There are also infrared sensors in front and behind the manipulator. When the sensor beam is broken, a relay turns off the hydraulic pump.

A start and stop relay is used to start and stop the hydraulic pump. However, this is not enough to stop the manipulator as hydraulic pressure is maintained in the accumulator. Another remote emergency stop button was also installed in the circuit between the FPGA and manipulator. When this circuit board emergency stop button is activated, all PWM signals to the manipulator are stopped thus closing all actuator valves. This effectively stops the movement of the manipulator even when the accumulator is still fully charged.

Therefore, when the manipulator is required to stop instantly both stop buttons should be pressed simultaneously.

3.3 The Circuit Board

The manipulator was originally controlled by discrete electronics on the 8 A3 sized original PCBs. All of the onboard electronics have been unplugged. The function of the single new A5 sized circuit board is to amplify the FPGA board PWM signals for

the valves, produce the correct voltage for the encoder scan outputs, and provide a suitable buffer for the encoder signals to the computer.
The major inputs and outputs for the circuit board are as follows:

- 32 bit link to the computer
- 15 input bits, 13 bits for the encoder, leaving 2 for other inputs
- 6 output bits to scan the encoders
- 6 positive amplified PWM signals for each of the valves
- 6 negative amplified PWM signals for each of the valves
- A common return for PWM signals
- 24V power output (positive and earth)
- 15V output
- 5V output

3.4 The Absolute Grayscale Encoders

The encoders return an absolute value that defines where the joints are orientated or positioned. The encoders consist of an encoder disk, a light source, a photocell cathode and related circuitry for each of the available 15 bits. The disk has 15 concentric rings of grayscale-patterned slots across its surface. The 15 photocells are able to sense light transmitted through the 15 rings, and send 15 bits for the light pattern sensed. Of the 15 available signals, 4 encoders only use 13 bits, while the other 2 use 14 bits.

There are 6 encoders, these encoders are multiplexed so that only one encoder is read at a time. Therefore, only 20 wires (14 bits + 6 select lines) are needed instead of 84 wires. Each encoder has a 15-bit output that joins up to the same wires from the other encoders. Only one encoder is active driving the sensed bits down the wires at any one time. The active encoder is selected sequentially so that 6 select and read operations are required to determine the manipulator position.

3.5 Kinematics of Unimate Model 2000B

The Unimate model 2000B is a serial link robot. It has six degrees of freedom, with five of these being revolute joints and one being a prismatic joint. The Unimate Model 2000B consist of an arm segment and a wrist segment. The arm segment is made out of the Rotary, Down-Up and In-Out joints with the last being a prismatic joint. The wrist segment is made out of the Bend, Yaw and Swivel joints.

The arm segment is used to position the end-effector of the manipulator to a certain point (x, y, z) in the workspace. The wrist segment is then used to rotate the end-effector into a certain orientation (γ, β, α). It is not coincidental that the robot has been designed with six degrees of freedom - six degrees of freedom is the minimum requirement for a manipulator to be able to move its end-effector to any point and orientation within the manipulator workspace

The manipulator kinematics have been described using the Denavit-Hartenberg notation. The Denavit–Hartenberg description of the robot will yield 6 matrices of the form shown in equation **(3-1)**, where the matrix A_i^{i-1} represents the position and orientation of the frame i relative to frame i-1. The first three 3x1 column vectors of A_i^{i-1} contain the direction cosines of the coordinate axes of frame i, while the last 3x1 column vector represents the position of the origin O_i. In other words, the former represents the rotation of frame X^i to frame X^{i-1} and the latter represents the translation of frame X^i to frame X^{i-1}. The 4^{th} row of the matrix augments it into a square matrix form, allowing it to be multiplied by other joint matrices to provide a similar homogeneous transfer matrix across multiple joints.

$$A_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-1)}$$

The forward kinematics for the Unimate 2000B have been derived in previous work [Yang 2000]. Equations **(3-3)** to **(3-8)** show the Denavit–Hartenberg matrices for this particular robot. These matrices were developed using equation **(3-1)** and the

data in Table 1. Equations **(3-3)** to **(3-8)** can be combined to find the relative relationship between any of the parameters, ie the position and orientation of the n^{th} joint relative to the base frame could be found from equation **(3-2)**.

$$A_n^0 = A_1^0 A_2^1 \ldots A_{n-1}^{n-2} A_n^{n-1} \qquad \textbf{(3-2)}$$

$$A_1^0 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-3)}$$

$$A_2^1 = \begin{bmatrix} -\sin\theta_2 & 0 & \cos\theta_i & -a_2\sin\theta_2 \\ \cos\theta_2 & 0 & \sin\theta_2 & a_2\cos\theta_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-4)}$$

$$A_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-5)}$$

$$A_4^3 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-6)}$$

$$A_5^4 = \begin{bmatrix} \cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ \sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-7)}$$

$$A_6^5 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(3-8)}$$

Table 1 Denavit Hartenburg Constants for the Unimate 2000B.

Link number	**α_i (degrees)**	**a_i (mm)**	**d_i (mm)**	**θ_i (degrees)**
1	+90	0	1066.8	θ_1
2	+90	120.65	0	θ_2
3	-90	0	d_3	0
4	90	0	0	θ_4
5	-90	0	200 (L_b)	θ_5
6	0	0	440 (L_y)	θ_6

3.6 Inverse Kinematics

The direct kinematics equation, establishes the functional relationship between the joint variables and the end-effector position and orientation. The inverse kinematics consist of the determination of the joint variables corresponding to a given end-effector position and orientation. The solution of this problem is fundamental to transform the motion specification, assigned to the end-effector in the operation space, into the corresponding joint space motions that allow execution of the desired motion.

The Unimate 2000B has six degrees of freedom therefore it has a finite number of solutions for the inverse kinematics. A manipulator arm must have at least six degrees of freedom in order to locate its end-effector at an arbitrary position with an arbitrary orientation in its workspace. Manipulator arms with less than 6 degrees of freedom are not able to perform such arbitrary positioning. However if a manipulator arm has more than 6 degrees of freedom, there exist as infinite number of solutions to the inverse kinematics equations and solutions that avoid the mechanism singularities are usually chosen.

The 6 inverse kinematics equations are shown below, the derivation of which can be found in Appendix A.

$$\theta_1 = \tan^{-1}\left(\frac{r_{23} - \frac{p_y}{Ly}}{r_{13} - \frac{p_x}{Ly}}\right)$$

$$\theta_6 = \tan^{-1}\left(\frac{c_1 r_{22} - s_1 r_{12}}{s_1 r_{11} - c_1 r_{21}}\right)$$

$$\theta_2 = 2 \times \tan^{-1}(t)$$

Where

$$t = -(2k_2) \pm \frac{\sqrt{4k_2^2 - 4(k_1 + a_2)(a_2 - k_1)}}{2(k_1 + a_2)}$$

$$k_1 = (Lb \times r_{11}s_6 + Lb \times r_{12}c_6 + p_x - Ly \times r_{13})/(c_1)$$

$$k_2 = (Lb \times r_{21}s_6 + Lb \times r_{22}c_6 + p_y - Ly \times r_{23})/(s_1)$$

$$\theta_5 = \tan^{-1}\left(\frac{r_{32}s_6 - r_{31}c_6}{r_{33}}\right)$$

$$d_3 = \frac{k_1 + a_2 s_2}{c_2}$$

$$\theta_4 = A\tan 2\left(\frac{(-r_{31}s_6 - r_{32}c_6)}{(c_5 c_6 r_{31} - c_5 s_6 r_{32} - r_{33}s_5)}\right) - \theta_2$$

3.6.1 Multiple Solutions to Inverse Kinematics

The six degrees of freedom manipulator has a fixed number of joint solutions to any given end-effector position and orientation. The solution of the inverse kinematics can be simplified by setting the following restrictions on the robot configuration:

- Restrict θ_5 to $-\pi/2 \le \theta_5 \le \pi/2$
- Restrict θ_6 to $-\pi \le \theta_6 \le \pi$

3.7 The Jacobian

The Jacobian matrix is used for mapping joint velocities to end effector frame velocities, and vice versa. The Jacobian is used in this project for controlling the robot using the joystick and calculating the error in the paths followed.

Let the Jacobian be defined as follows:

$$\dot{p} = J_P(q)\dot{q} \quad \textbf{(3-9)}$$

$$\omega = J_0(q)\dot{q} \quad \textbf{(3-10)}$$

Where

J_p is the (3 x n) matrix relative to the contribution of the joint velocities $\dot{q}$ to the end effector linear velocity $\dot{p}$.

J_0 is the (3 x n) matrix relative to the contribution of the joint velocities $\dot{q}$ to the end effector angular velocity ω .

This can be rewritten as

$$v = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J(q)\dot{q} \quad \textbf{(3-11)}$$

Where J is the manipulator geometric Jacobian:

$$J = \begin{bmatrix} J_P \\ J_0 \end{bmatrix}$$

The Jacobian J can be calculated as follows, given the generic structure of the Jacobian as:

$$J = \begin{bmatrix} j_{P1} & \cdots\cdots & j_{Pn} \\ j_{01} & \cdots\cdots & j_{0n} \end{bmatrix}$$

Following rules 1 & 2 below the Jacobian for the Unimate 2000B can be constructed:

Rule 1: If joint i is prismatic then $J_{0i} = 0$, and $J_{Pi} = z_{i-1}$

Rule 2: If joint i is revolute then $J_{0i} = z_{i-1}$, and $J_{Pi} = z_{i-1} \times (p-p_{i-1})$

Equations **(3-12)** to **(3-47)** are the 36 elements of the 6 x 6 Jacobian matrix for the Unimate Robot. The Jacobian was derived using the Matlab symbolic toolbox, and the program shown in appendix C.

J[1,1]= -Ly*s5*s1*s2*c4-Ly*s5*s1*c2*s4+Ly*c1*c5+Lb*s1*s2*s4-Lb*s1*c2*c4-s1*c2*d3+a2*s1*s2 **(3-12)**

J[1,2]= -Ly*c1*s5*s2*s4+Ly*c1*s5*c2*c4-Lb*c1*c2*s4-Lb*c1*s2*c4-c1*s2*d3-a2*c1*c2 **(3-13)**

J[1,3]= c1*c2 **(3-14)**

J[1,4]= -Ly*c1*s5*s2*s4+Ly*c1*s5*c2*c4-Lb*c1*c2*s4-Lb*c1*s2*c4 **(3-15)**

J[1,5]= Ly*s1*s5+Ly*c5*c1*c2*s4+Ly*c5*c1*s2*c4 **(3-16)**

J[1,6]= 0 **(3-17)**

J[2,1]= Ly*s5*c1*s2*c4+Ly*s5*c1*c2*s4+Ly*s1*c5-Lb*c1*s2*s4+Lb*c1*c2*c4+c1*c2*d3-a2*c1*s2 **(3-18)**

J[2,2]= -Ly*s1*s5*s2*s4+Ly*s1*s5*c2*c4-Lb*s1*c2*s4-Lb*s1*s2*c4-s1*s2*d3-a2*s1*c2 **(3-19)**

J[2,3]= s1*c2 **(3-20)**

J[2,4]= -Ly*s1*s5*s2*s4+Ly*s1*s5*c2*c4-Lb*s1*c2*s4-Lb*s1*s2*c4 **(3-21)**

J[2,5]= Ly*c5*s1*c2*s4+Ly*c5*s1*s2*c4+Ly*c1*s5 **(3-22)**

J[2,6]= 0 **(3-23)**

J[3,1]= 0 **(3-24)**

J[3,2]= Ly*s5*s2*c4+Ly*s5*c2*s4-Lb*s2*s4+Lb*c2*c4+c2*d3-a2*s2 **(3-25)**

J[3,3]= s2 **(3-26)**

J[3,4]= Ly*s5*s2*c4+Ly*s5*c2*s4-Lb*s2*s4+Lb*c2*c4 **(3-27)**

J[3,5]= Ly*c5*s2*s4-Ly*c5*c2*c4 **(3-28)**

J[3,6]= 0 **(3-29)**

J[4,1]= 0 **(3-30)**

J[4,2]=	s1	**(3-31)**
J[4,3]=	0	**(3-32)**
J[4,4]=	s1	**(3-33)**
J[4,5]=	-c1*s2*s4+c1*c2*c4	**(3-34)**
J[4,6]=	s5*c1*s2*c4+s5*c1*c2*s4+s1*c5	**(3-35)**
J[5,1]=	0	**(3-36)**
J[5,2]=	-c1	**(3-37)**
J[5,3]=	0	**(3-38)**
J[5,4]=	-c1	**(3-39)**
J[5,5]=	-s1*s2*s4+s1*c2*c4	**(3-40)**
J[5,6]=	s5*s1*s2*c4+s5*s1*c2*s4-c1*c5	**(3-41)**
J[6,1]=	1	**(3-42)**
J[6,2]=	0	**(3-43)**
J[6,3]=	0	**(3-44)**
J[6,4]=	0	**(3-45)**
J[6,5]=	c2*s4+s2*c4	**(3-46)**
J[6,6]=	s5*s2*s4-s5*c2*c4	**(3-47)**

Where $si = \sin(\theta_i)$

$ci = \cos(\theta_i)$

4 Controller Design

The problem of motion control of a manipulator will be addressed in this chapter, specifically a joint space control technique will be discussed. A joint space control approach has been implemented rather than an operational space control scheme. This prevents the inverse kinematics being solved at run time in the ISR, thus making the control algorithm less computationally expensive.

4.1 System Model

The Robot is a coupled system, hence the movements of one axis has an effect on the other axis. The 3 major axis (Up-Down, In-Out, and Rotary Motion) are the most heavily coupled. An approximate model of the robot arm considering only these motions can be seen in figures 4.1, 4.2, and 4.3.

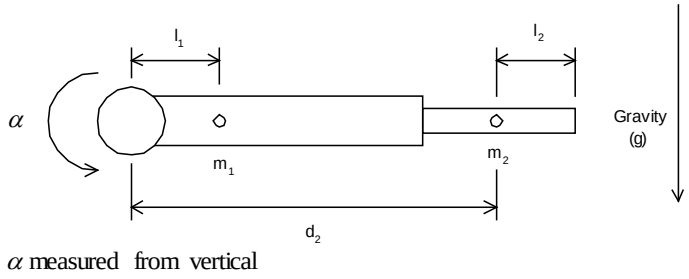

Figure 4-1 Up/Down Axis Side Elevation

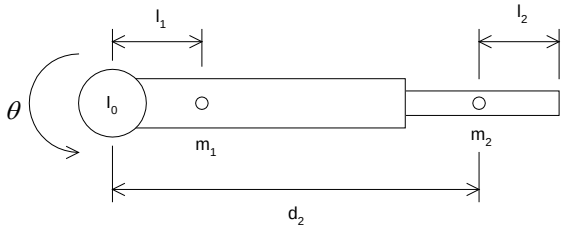

Figure 4-2 Rotate Left/Right Plan View

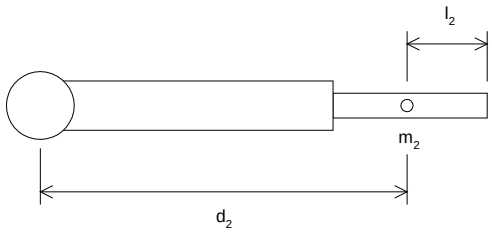

Figure 4-3 In/Out Side Elevation

Solving the equation of motion on these models allows the major robot axis to be described by equations **(4-1)** to **(4-3)**.

The Rotary Motion:

$$\tau = \left(m_1 l_1^2 + I_0 + I_1 + I_2 + m_2 d_2^2\right)\ddot{\theta} + 2m_2 d_2 \dot{\theta}_1 \dot{d}_2 \qquad \textbf{(4-1)}$$

Up/Down Motion:

$$\tau = \left(m_1 l_1^2 + I_1 + I_2 + m_2 d_2^2\right)\ddot{\alpha} + 2m_2 d_2 \dot{\alpha}\dot{d}_2 + \left(m_1 l_1 + m_2 d_2\right) g \cos(\alpha) \qquad \textbf{(4-2)}$$

In/Out Motion:

$$F = m_2 \ddot{d} - m_2 d_2 \dot{\alpha}^2 + m_2 g \sin(\alpha) + m_2 \ddot{\theta} - m_2 d_2 \dot{\theta}^2 \qquad \textbf{(4-3)}$$

where l_1, l_2, m_1, I_0, m_1, m_2, α, θ, d_2 are as described in figures 4-1 to 4-3,
g is gravity, and
I_1 and I_2 are:

$$I_1 = \frac{m_1 l_1^{\,2}}{3} \qquad \textbf{(4-4)}$$

$$I_2 = \frac{m_2 d_2^{\,2}}{3} \qquad \textbf{(4-5)}$$

These equations show the coupling between the axes, this can be seen better in Figure 4-4. The dashed lines in Figure 4-4 show the coupling between the three main axis

(In/Out, Up/Down, Rotate Left/Right). Treating the coupling as a disturbance allows the plant to be modelled as shown in Figure 4-5. If the three end-effector axes are also treated as uncoupled systems subject to a disturbance then the robot can be treated as a system of 6 SISO systems.

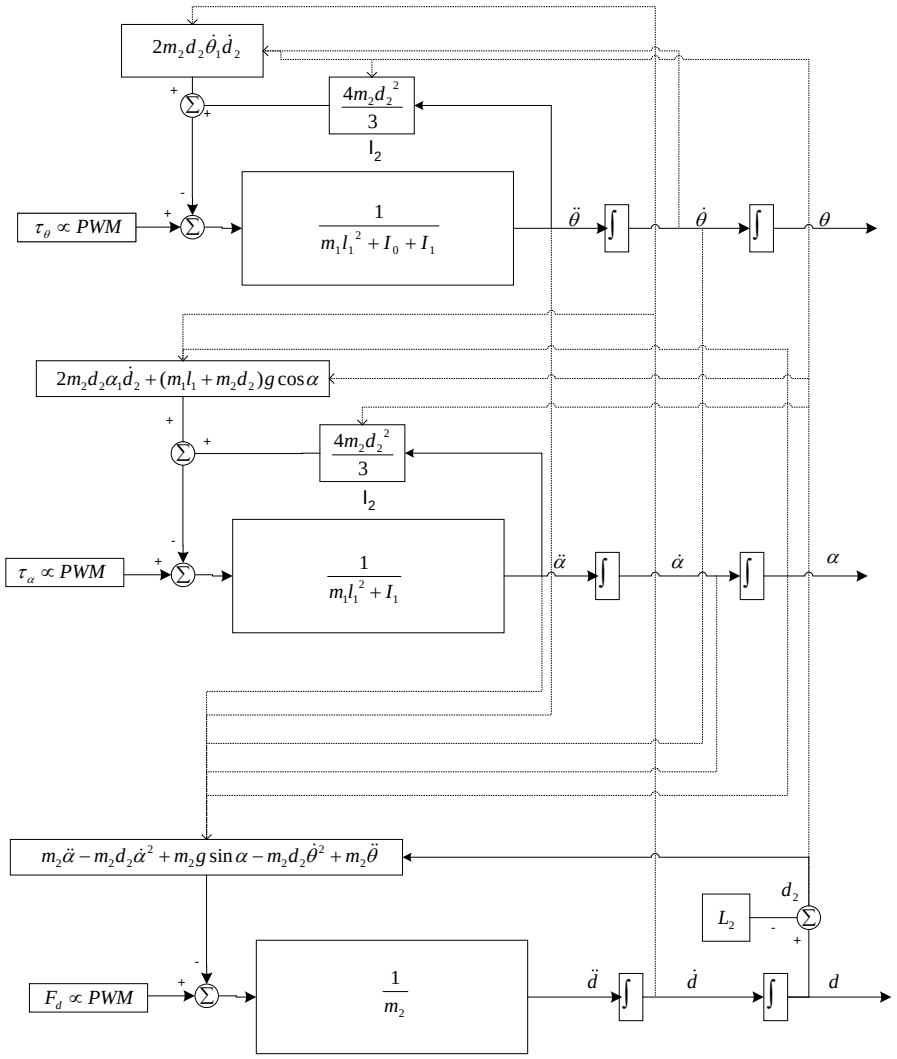

Figure 4-4 Plant Model

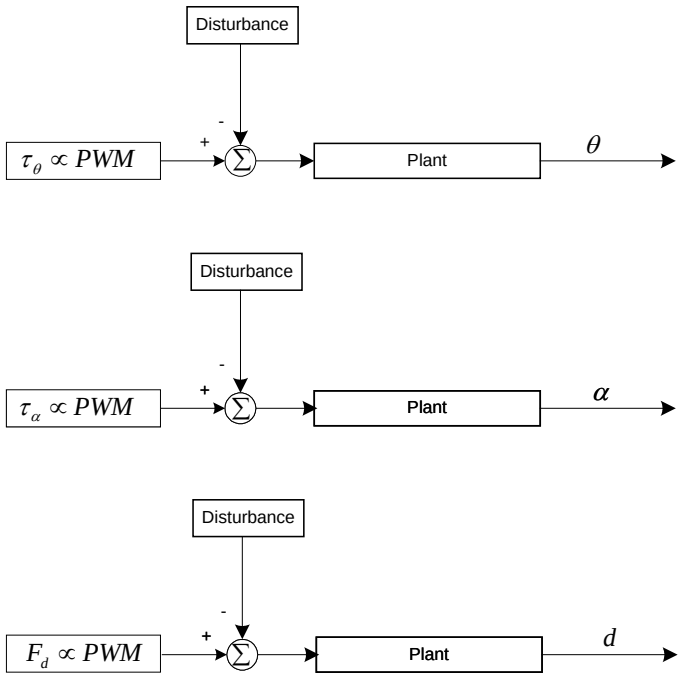

Figure 4-5 Simplified Model of Plant

If the system is to be modelled this way, then the controller must have good disturbance rejection qualities.

4.2 PI Control

PI control is used for the control algorithm as it allows a generic control structure to be used for all 6 of the SISO systems. Figure 4-6 shows the arrangment of a PI controller.

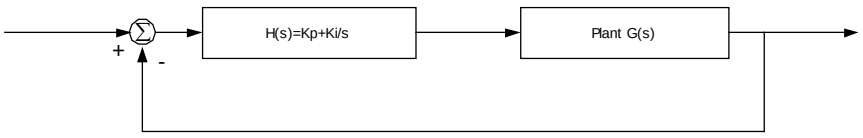

Figure 4-6 PI Controller (Frequency Domain)

However the actual implementation of the controller will not be continuous as in Figure 4-6, therefore the digital version of the control algorithm shown in Figure 4-7

is required (NB the Δ represents a delay equal to the control system time sampling interval).

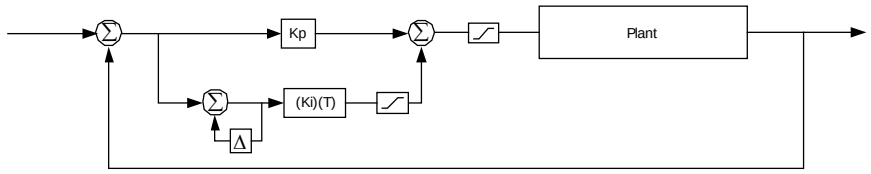

Figure 4-7 PI Controller (Digital Implementation)

4.3 Frequency Response of System

The problem with a model based control design is the lack of knowledge of any of the model parameters. The approach that has been taken in the design of this controller is to use a frequency response method. This allows the robot to be treated as a series of 'black box' SISO systems, with each having a PWM input, and a position output. The advantage of doing this is that the measured data can be displayed on a bode plot, allowing the use of classical loop shaping techniques to design a controller. Figures 4-8 through to 4-15 show the Bode plots for the robot. The figures also show the frequency characteristics of the controller and the development of these curves is discussed further in section 4.4.

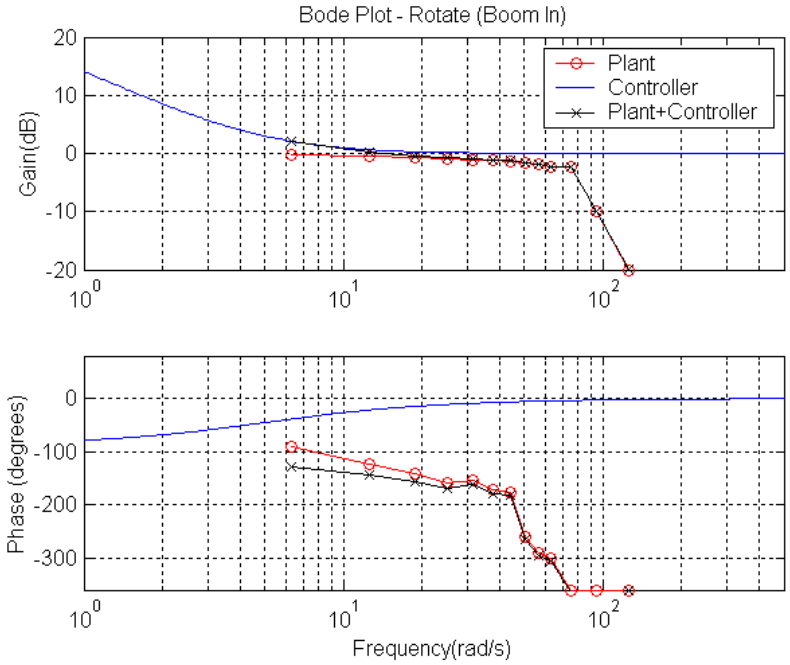

Figure 4-8 Bode Plot - Rotate (Boom In)

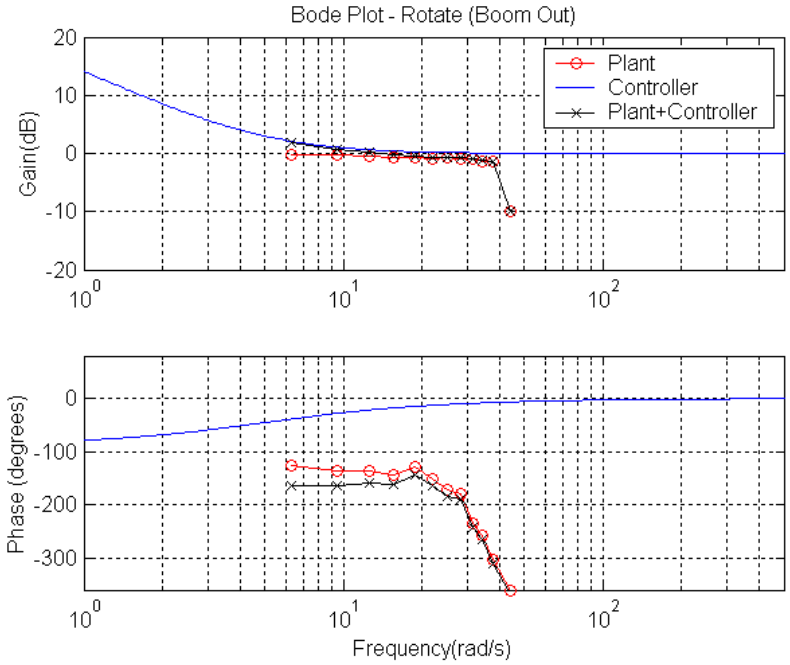

Figure 4-9 Bode Plot - Rotate (Boom Out)

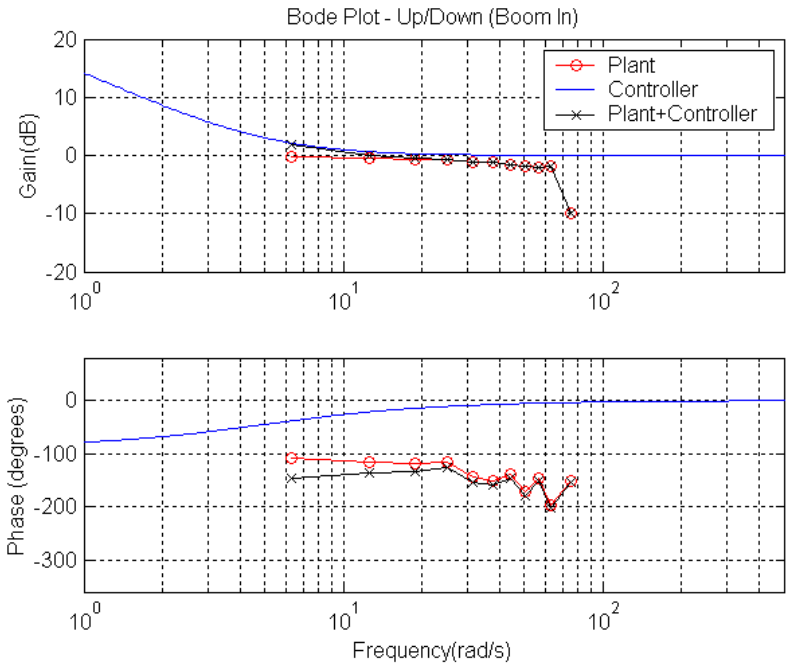

Figure 4-10 Bode Plot - Up/Down (Boom In)

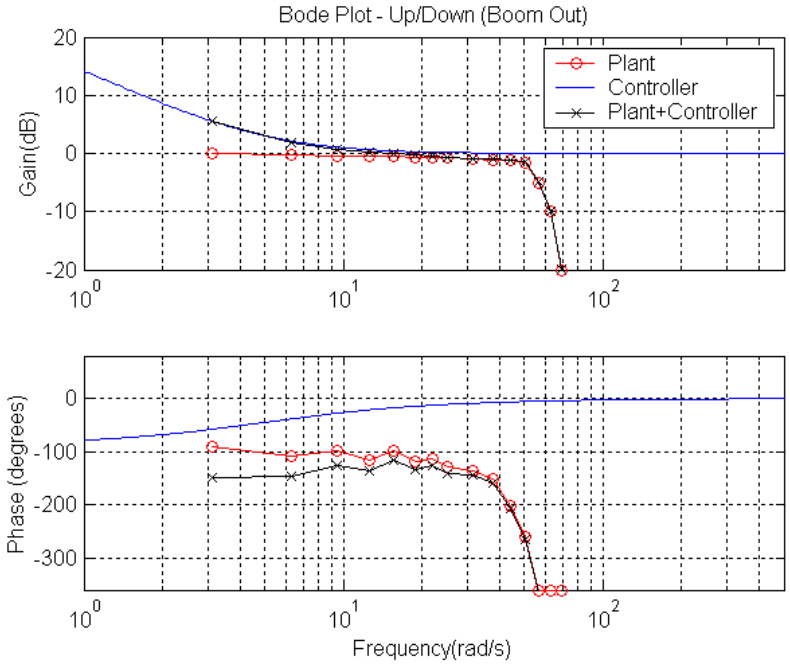

Figure 4-11 Bode plot - Up/Down (Boom Out)

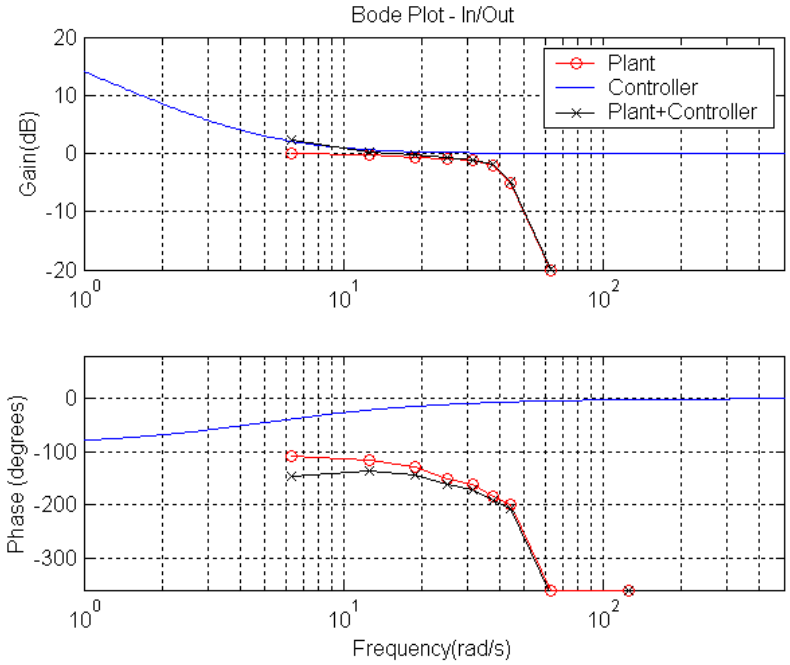

Figure 4-12 Bode Plot - In/Out

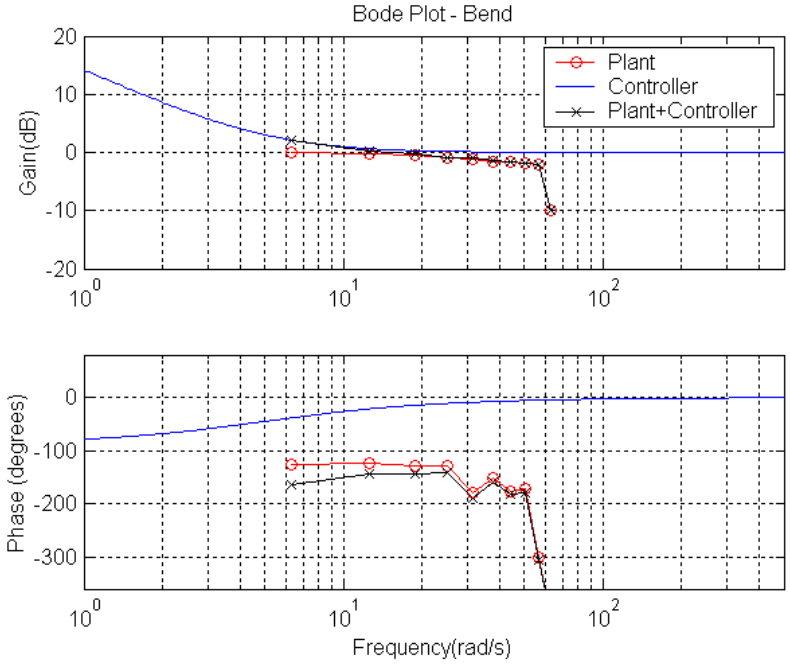

Figure 4-13 Bode Plot - Bend

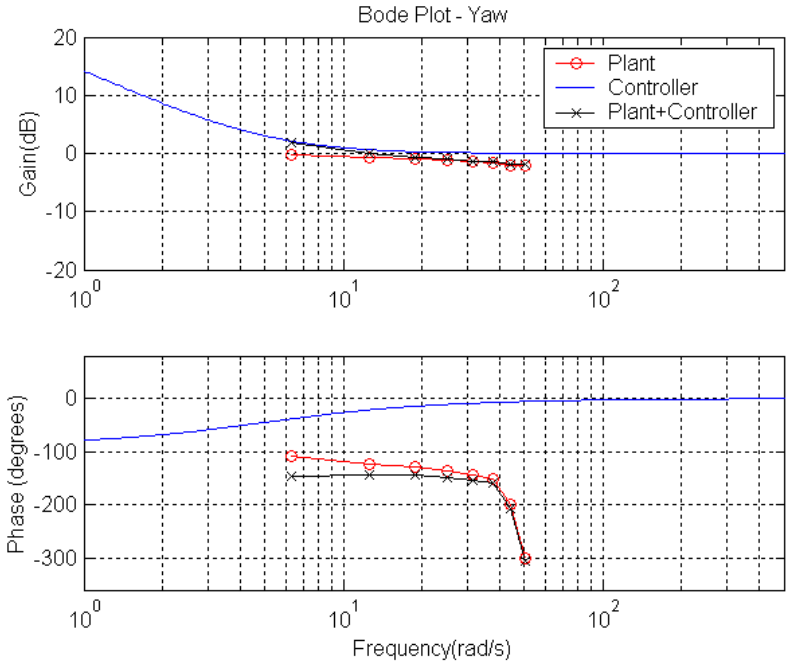

Figure 4-14 Bode Plot - Yaw

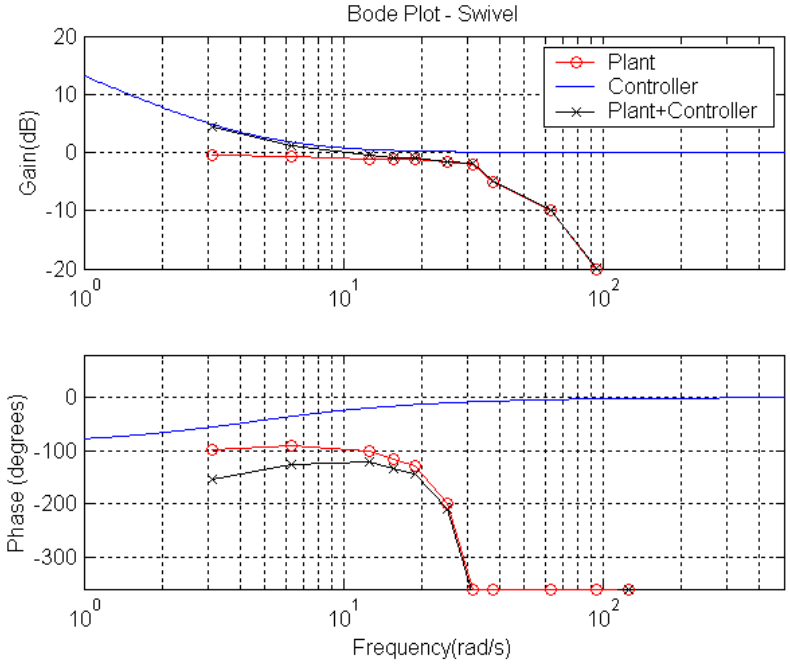

Figure 4-15 Bode Plot - Swivel

4.4 Controller Design

The PI controllers have been designed with the following driving factors:

- Low Bandwidth (Bandwidth = 10 rad/s)
- High Gain at Low Frequencies
- Conservative Gain and Phase Margins

The controller with these attributes has excellent tracking properties, with the high gain at low frequencies forcing the steady state error $\rightarrow 0$. The conservative gain and phase margins ensure greater stability for realistic loads. Figures 4-8 through to 4-15 show the frequency response of the controllers, as well as the corresponding frequency response of the plant and controller combined.

	Kp	**Ki**
Rotate	1	5
Up/Down	1	6
In/Out	1	5
Bend	1	7
Yaw	1	6
Swivel	5	6

Table 2 Parameters for PI Controller with a system sample time of 1ms

4.5 Performance of Controller

4.5.1 Uncoupled Ramp Response

The ramp response for each axis can be seen in figures 4-16 through to 4-21. These results show that the controllers work very well at tracking a ramp, with a near 0 steady state error in all cases except where the speed of the ramp is exceeds that possible for the actuator.

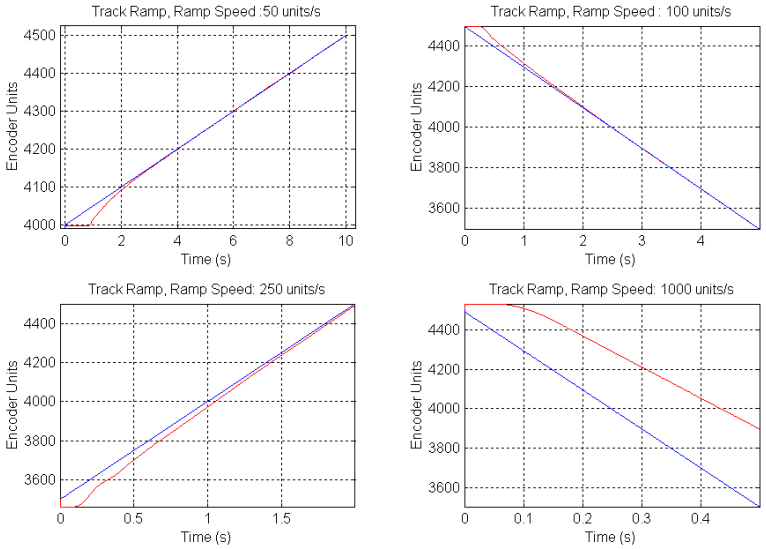

Figure 4-16 Rotate Ramp Response

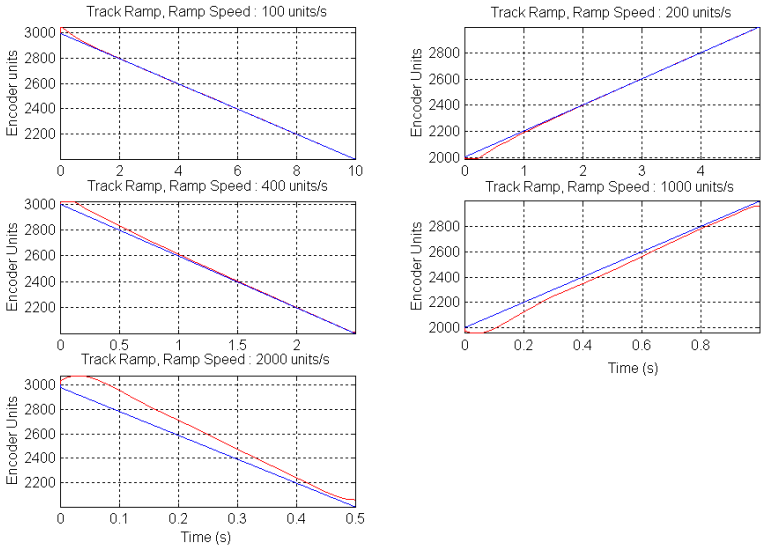

Figure 4-17 Up/Down Ramp Response

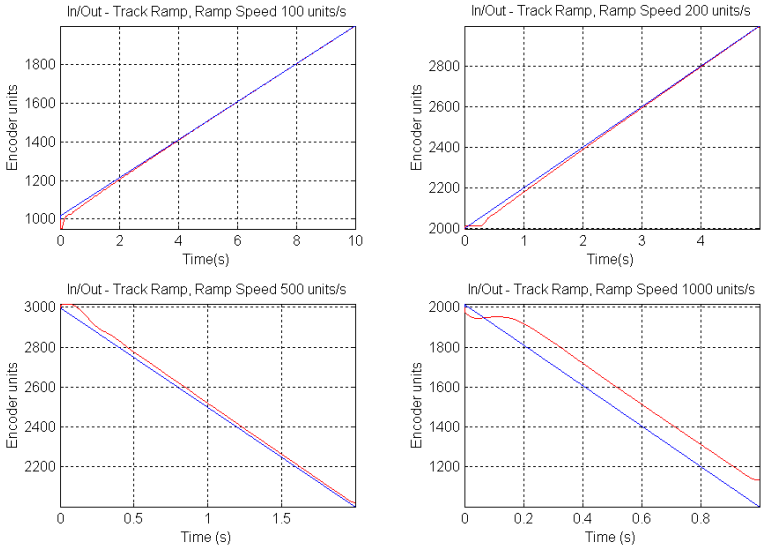

Figure 4-18 In/Out Ramp Response

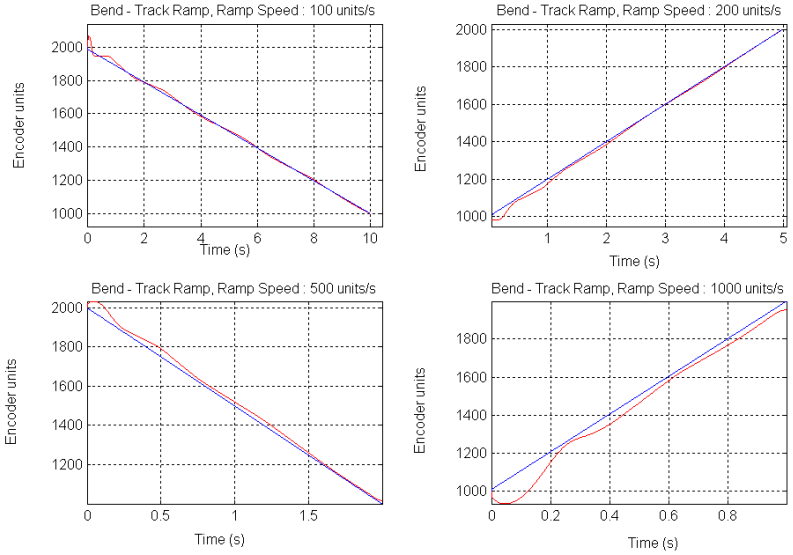

Figure 4-19 Bend Ramp Response

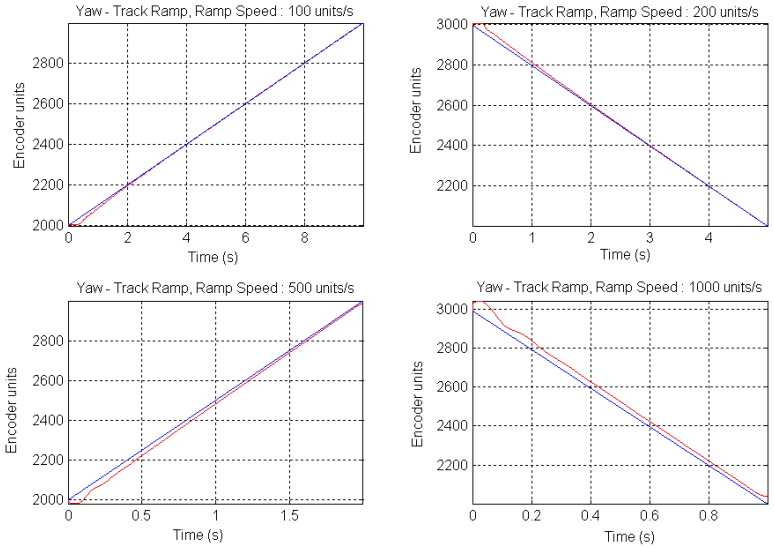

Figure 4-20 Yaw Ramp Response

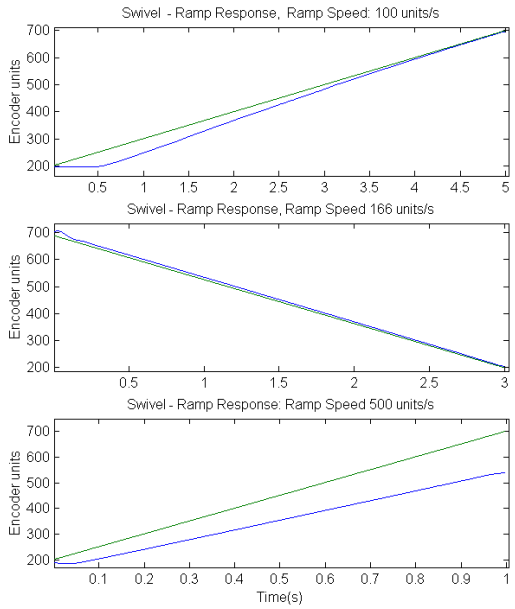

Figure 4-21 Swivel Ramp Response

4.5.2 Coupled Ramp Response

The ramp responses for the joints when coupling effects are present are shown in figures 4-22 to 4-26. The coupled response is not shown for the swivel axis as no significant coupling forces are present. The coupled response was tested by setting each joint to follow a ramp simultaneously for a period of 2s. The ramp speed chosen for each joint was the maximum attainable while still allowing good tracking accuracy, these ramp speeds were derived form the uncoupled ramp responses in the previous section. The figures show that the controllers used suppress the effect of the coupling forces and allow each joint to track the ramps accurately.

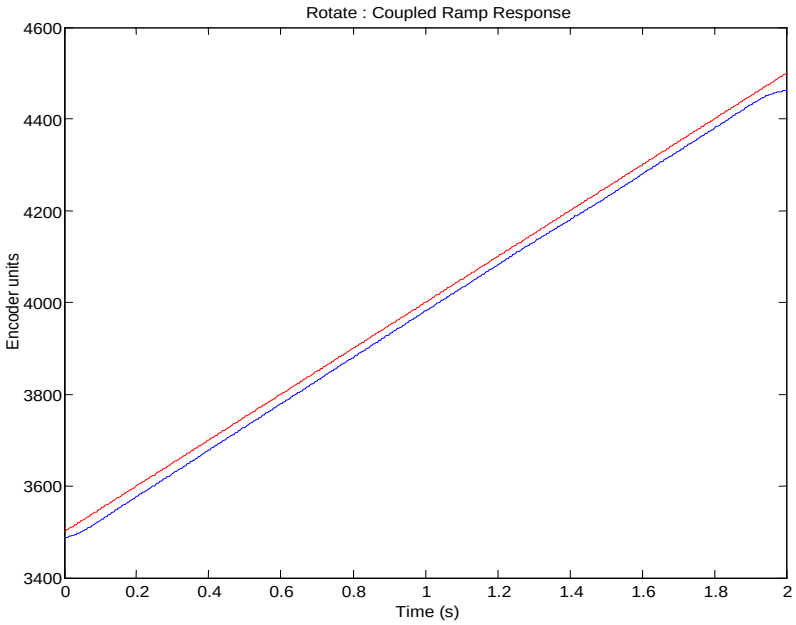

Figure 4-22 Rotate Coupled Ramp Response

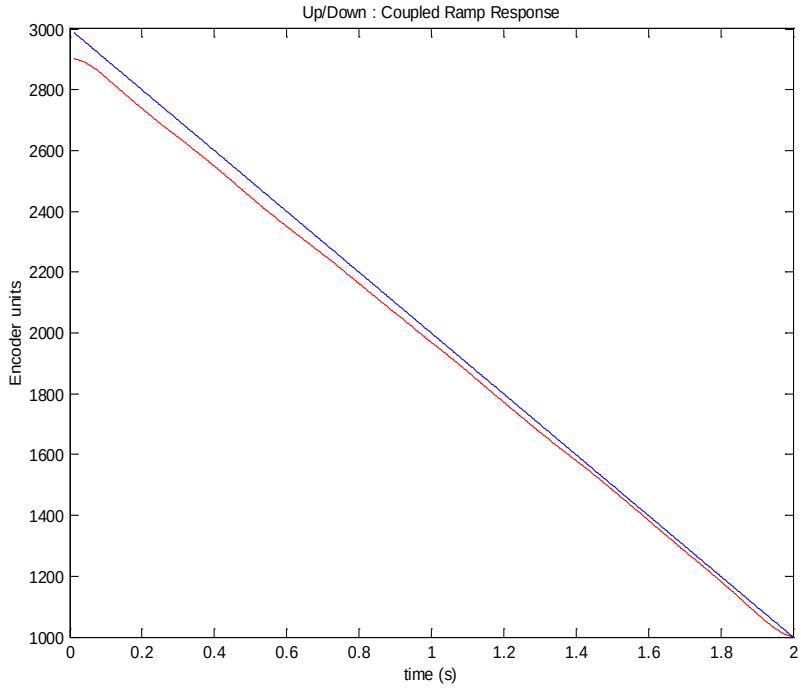

Figure 4-23 Up/Down Coupled Ramp Response

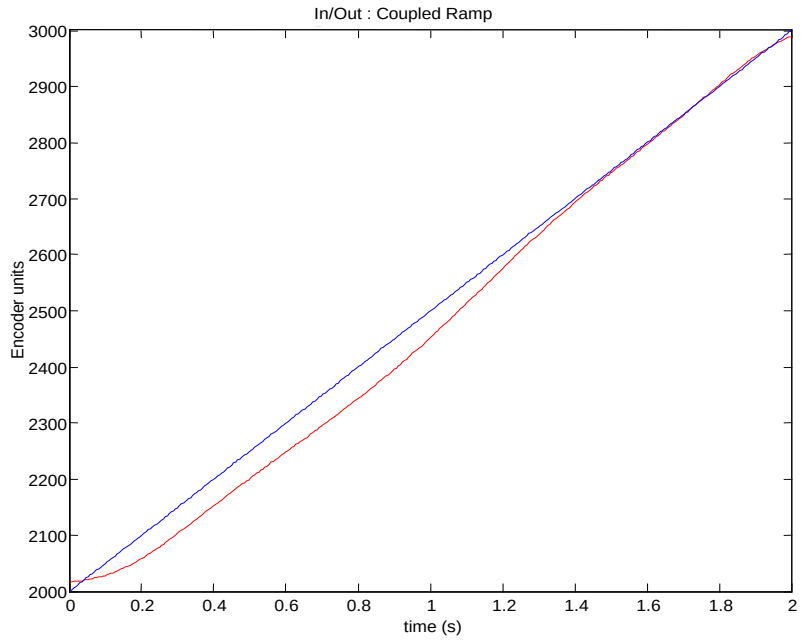

Figure 4-24 In/Out Coupled Ramp Response

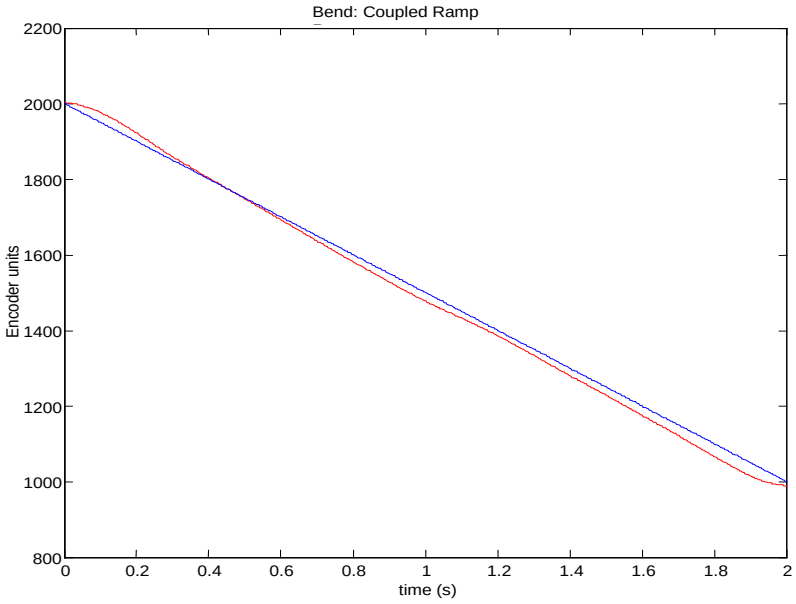

Figure 4-25 Bend Coupled Ramp Response

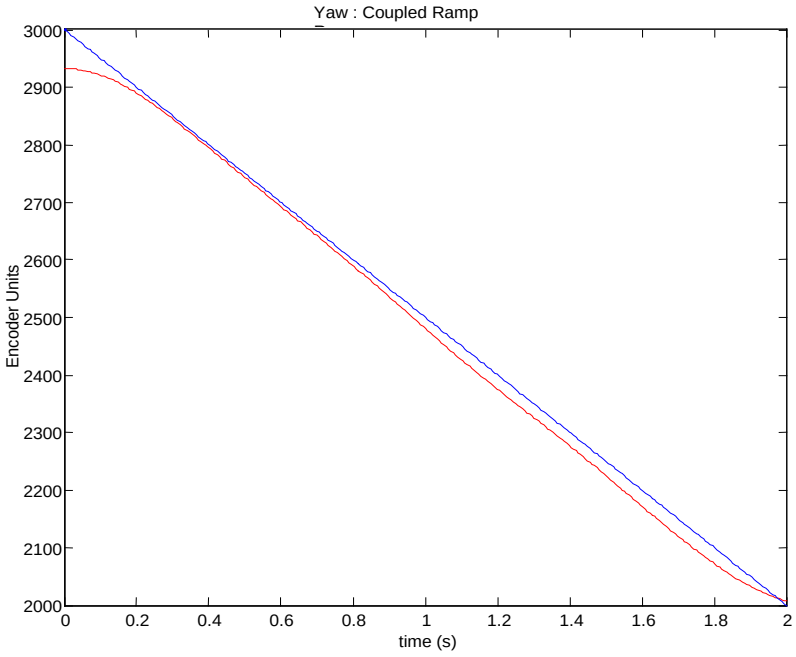

Figure 4-26 Yaw Coupled Ramp Response

5 Software Design

The Software can be split into two distinct sections, Real Time software, and non real time software (see appendix G for user instructions). All software has been developed using the Microsoft Visual Studio environment. This IDE (Integrated Development Environment) allows simple creation and management of object orientated C++ code. Real Time software is compiled into RTSS files, the GUI win32 process is a standard .exe file. An added advantage of using a development environment such as Visual Studio is the ability to debug processes at run time. Unlike win32 processes an RTSS process cannot be debugged using this method. However the RTSS program can be compiled into a win32 process easily by changing a few compiler and Visual Studio environment settings. Doing this allows the process to be debugged using the Visual Studio environment.

5.1 Real Time Software

The design of real time software requires careful consideration of a number of factors, most importantly timing issues. This application makes use of a single ISR, and hence is not very complicated, the main issue here is reducing the interrupt latency, and ensuring the ISR is as efficient as possible. Because a Pentium processor is being used in this implementation cache flushing can be a major cause of lengthy interrupt latencies. While this cannot be prevented careful design can reduce the problems.

5.1.1 Cache Memory

Fast memory can be quite expensive, so it is common to purchase masses of cheap (but slow) memory, and a relatively small amount of very fast *cache* memory. The idea is that the memory management processor, as a background activity, keeps filling the instruction and data caches with the information that it expects to be required next. If the required information is indeed found in cache memory when required, then this is called a cache-hit, otherwise a cache miss occurs. A time-critical instruction loop will run fastest if all its instructions fit and remain within the instruction cache.

5.1.2 How Cache Works

Caches are introduced into a system to buffer the mismatch between main memory and processor speeds. The cache is designed so that its access time matches the processor cycle time. Thus, if the processor is running with a 100MHz clock the L1 cache should be able to respond to a memory request in approximately 10ns. The typical size of these L1 caches is 8-16kb. Many system designs also include an off-chip cache, which is called the second-level cache or the L2 cache.

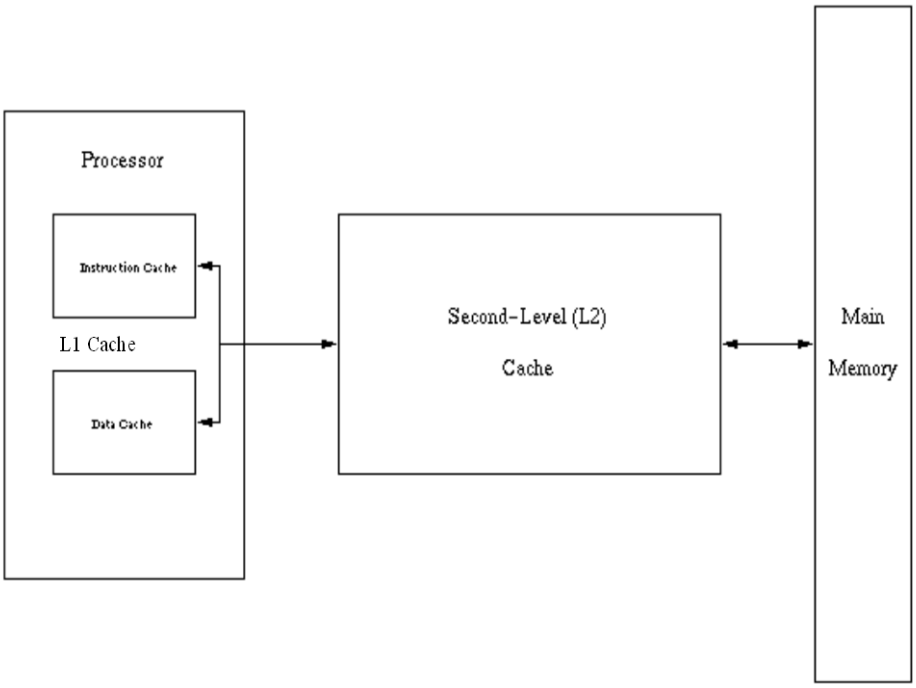

Figure 5-1 Model of Typical Cache Set-Up

The L2 cache can be anywhere from 128kb to 4Mb in size. The on-chip cache is called the first-level or primary cache. While the first-level cache must match the processor speed, the second-level cache can be somewhat slower, but not as slow as the main memory.

When the processor makes a memory request, the request first passes to the primary cache. If the data item is found in this cache, there is a cache *hit*. If the data item is not

found in the primary cache, there is a cache *miss* and the memory request is forwarded to the L2 cache. If the data item is found in this cache, there is an L2 cache hit and the data is passed back to the primary cache. If the data is not found in the L2 cache, the request is finally forwarded to the main memory. When the main memory responds to the memory request, the data item is passed back to the L2 cache and then the primary L1 cache.

Caches work well because the primary cache can usually service the memory request. The reason that the primary cache can handle so many memory requests has to do with two aspects of program behaviour:

Temporal Locality:
If a memory location is referenced, it is very likely that the memory location will be referenced again in the near future.

Spatial Locality:
If a memory location is referenced, it is very likely that a nearby memory location will also be referenced in the near future.

Spatial locality is embodied in a cache design by grouping sequential bytes of memory into a single *cache line*. Information transfer between the cache and the memory is in terms of complete cache lines, rather than individual bytes. Thus if the program needs a particular byte, the entire cache line containing that byte is obtained from the memory.

5.1.3 Design of code to Maximise Cache Hits

It is important that the critical part of the code (the innermost loops) fits in the code cache. Frequently used pieces of code or routines which are used together should preferably be stored near each other.

If the critical part of the code accesses big data structures or random data addresses, then all frequently used variables (counters, pointers, control variables, etc.) should be stored within a single contiguous block of max 4 Kbytes so that a complete set of cache lines are free for accessing random data.

The L1 data cache consists of 256 or 512 lines of 32 bytes each. For each read of a data item which is not cached, the processor will read an entire cache line from memory. The cache lines are always aligned to a physical address divisible by 32. When a byte is read at an address then the other 31 bytes in the cache line can be read or written to at almost no extra cost. Arranging data items that are used near each other together into aligned blocks of 32 bytes of memory can take advantage of this. If, for example, a loop accesses two arrays, then it is advantageous to interleave the two arrays into one array structure, so that data items that are used together are also stored together.

Separation of Read only and Read Write data structures is also a good idea. Data that has been modified will need to be written back to memory when removed from cache, whereas unmodified data blocks will not. Separation prevents the unnecessary write back of unmodified data contained in a cache line.

5.1.4 Cache and Problems with Real Time applications

While the cache memory designed into advanced processors can significantly speed up the average performance of many programs, it also causes performance variations that can surprise system designers and cause problems during product integration and deployment. The two main problems are lack of predictability and lack of determinacy.

For our purposes, predictability means that when you write a piece of code, it not only does what you expect, but takes the amount of time you expect it to take. Determinacy means that the results of executing a piece of code, including how long it takes to execute, don't change from run to run. With the advent of cache memories on Intel chips, determinacy and predictability have deteriorated compared to the older x86 chips. The problem with poor predictability is that performance of programs can be very difficult to characterise during development. The problem with poor determinacy is that a program can have latent timing problems that are extremely difficult to detect, reproduce, and correct.

With cache memory, code and data that are accessed frequently, especially small loops tend to be available in the fast cache memory most of the time. Code and data that are infrequently accessed tend not to be found in cache because they have been displaced by newer values, and these require lengthy accesses to main memory DRAM. The advantage of using cache is that frequently used memory locations are usually accessed quickly, which gives high average execution times. The disadvantage of cache is that speedups are of a statistical nature, and there are no guarantees for any particular short period of time.

The very worst execution times are instances where not only were there all cache misses during the executed code, but also DRAM refreshes or other untoward events occurred to further slow down the program. Execution time can vary wildly from run to run, depending on what other programs have done to the cache memory between ISR executions. It is possible to make the determinacy better by flushing the cache prior to the execution of the ISR, however the speed of the program will deteriorate to near the worst case delay time.

5.2 Communication between RTSS & win32 processes

Communication between the RTSS process and win32 processes is achieved through shared memory, and event objects (for synchronisation purposes). The following shared memory objects have been created:

- FIFO to send data from RTSS process to win32 process; data includes the current position, the desired position, and PWM of each valve.
- FIFO to send a desired path from win32 process to the RTSS process.
- A single data structure that controls the behaviour of the Real Time process

A FIFO (First in First Out) is a type of buffer that ensures that the first object written to the buffer is the first read out. The FIFO has been created with two write routines. The first routine continues writing to the buffer even when it is full, so that data is overwritten at the read end to make space for the new data. The other write routine does not write information to the FIFO if the buffer is full. Dealing with shared

memory objects is very simple using RTX, a shared memory object is created using the function call RtCreateSharedMemory, other processes can then use the shared memory object's given name using the call: RtOpenSharedMemory to map the shared memory object. RtCreateSharedMemory returns a handle and sets a location to the base address of the shared memory. Figure 2-1 shows a diagrammatic view of the IPC.

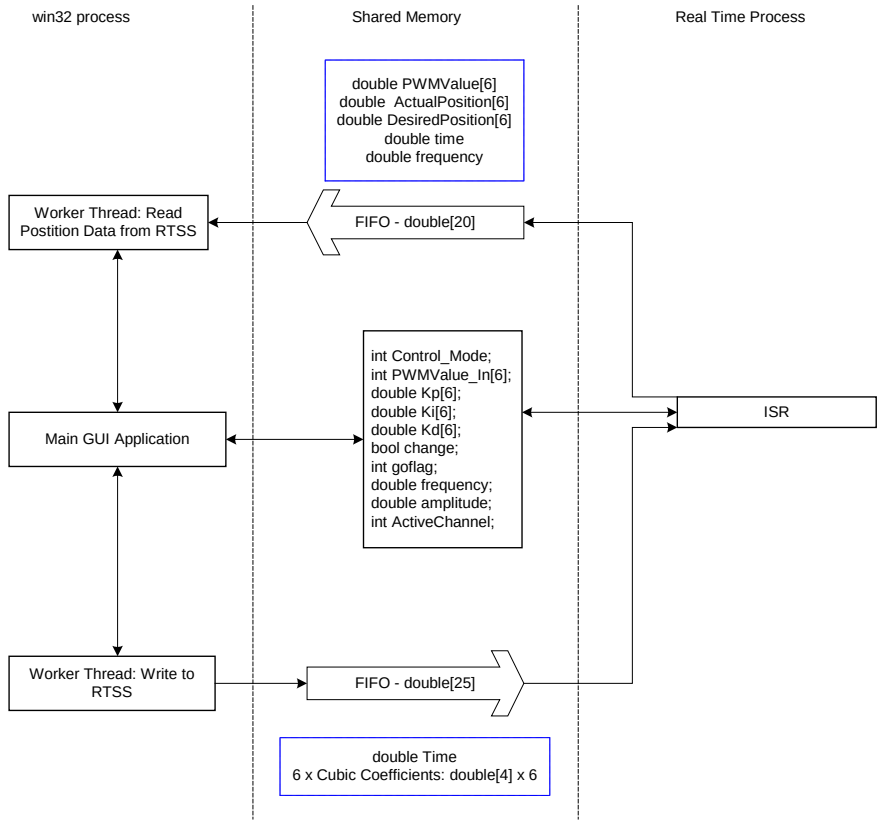

Figure 5-2 Inter-process Communication

5.3 Real Time Software

The Real Time software consists of two components, the main thread, and the ISR. The main thread initialises everything before going into an infinite 'while' loop, the ISR is then called at a rate of 6kHz. Figure 5-3 shows the general structure of the real time process.

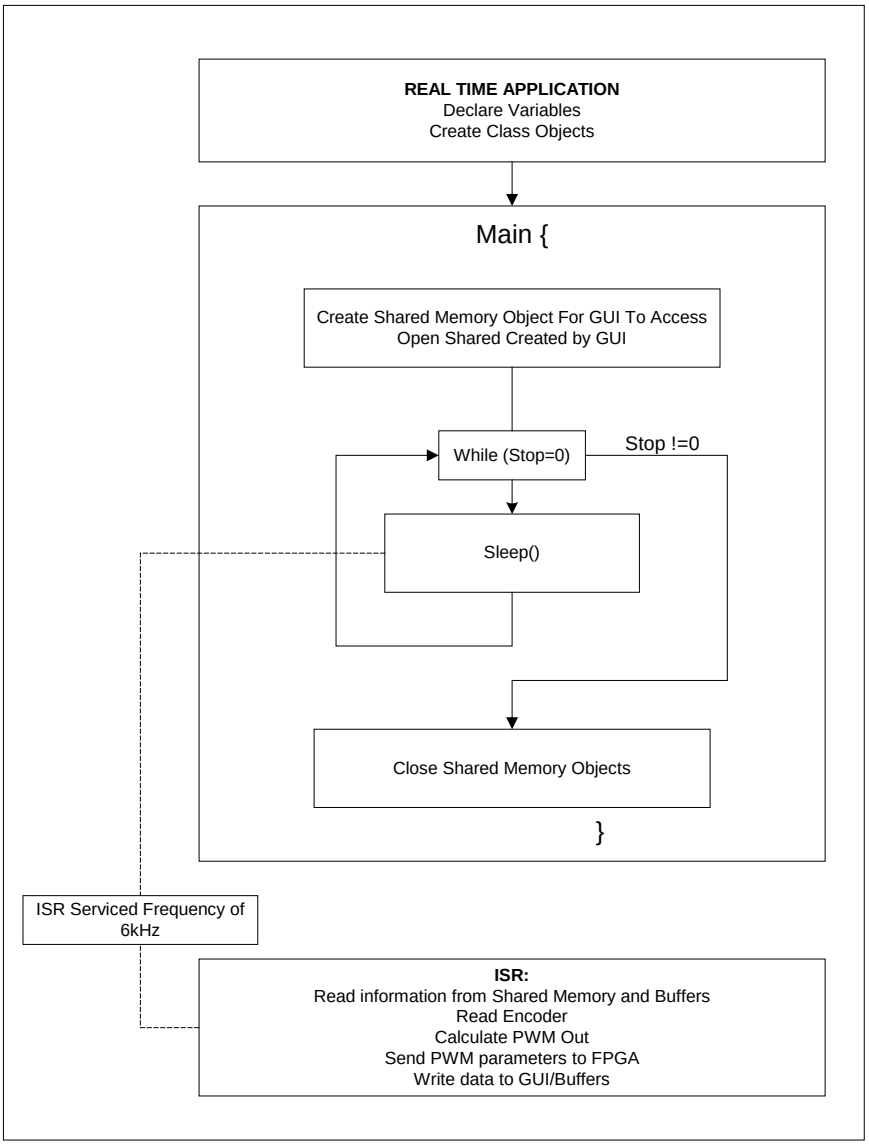

Figure 5-3 Layout of Real-Time Software Application

5.4 Graphical User Interfaces

The creation of the GUI is a major benefit of running a control system from a PC as the GUI allows easier user interaction with the system. Two Graphical User Interfaces have been created. One is used for controlling the robot, and the other for generating a path.

5.4.1 Main Control GUI

The main control GUI is shown in Figure 5-4. This GUI is responsible for controlling the behaviour of the robot. This arrangement allows the robot to run in a number of different modes:

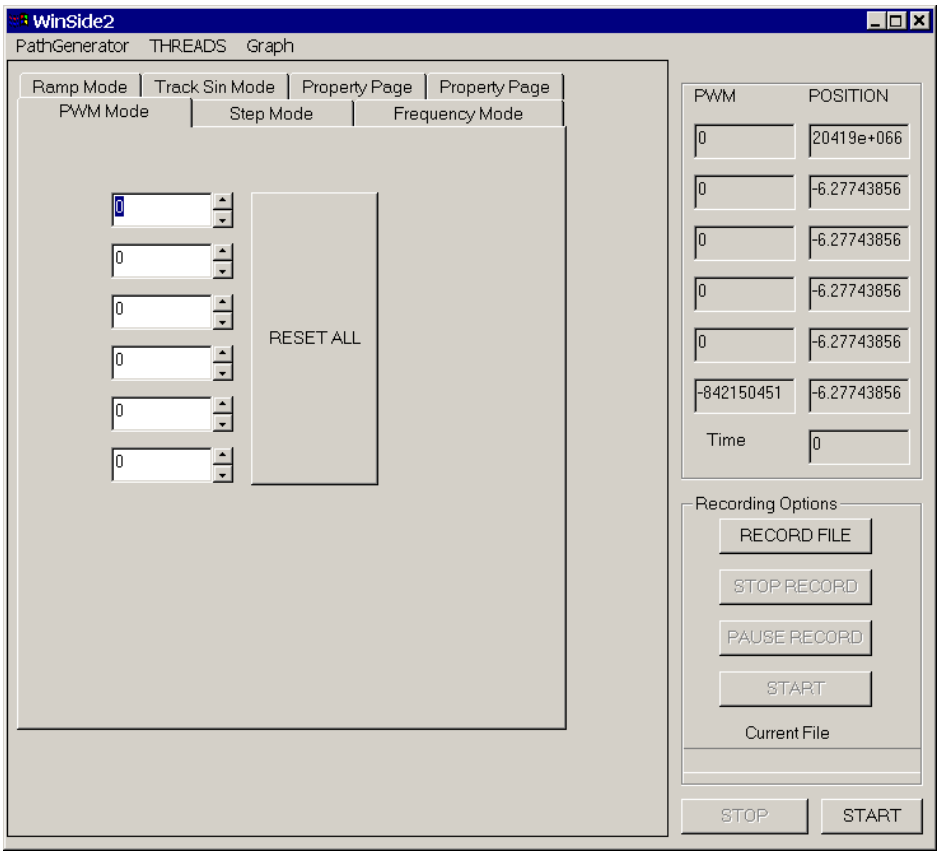

Figure 5-4 Main Robot Control GUI

1) *Direct PMW Control (Figure 5-5)*

Formatted

The PWM values for each of the 6 valves can be set directly. The PWM value is set through the use of spin boxes, with limits set between –100 and 100.

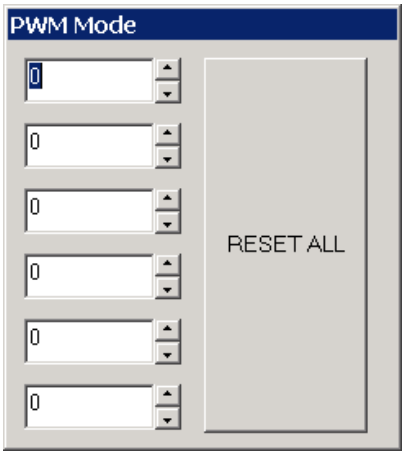

Figure 5-5 PWM Mode Property Page

2) *Sinusoidal PWM Control (Figure 5-6)*

Formatted

This mode is similar to the previous mode in that the user directly determines the PWM value, however a frequency (f) and amplitude (A) are chosen instead of a PWM value. The PWM is then calculated for each joint as follows:

$$PWM = A \times \sin(2 \times \pi \times f \times t)$$

This mode is particularly useful for determining the frequency response of the system.

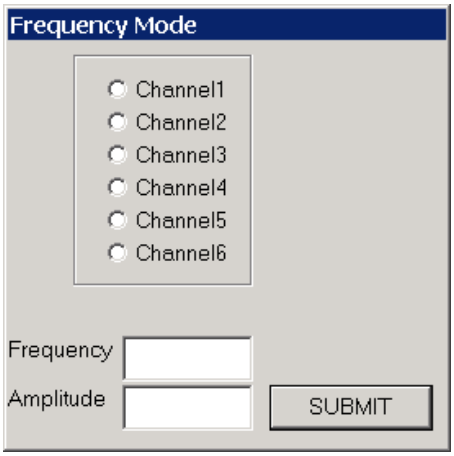

Figure 5-6 Frequency Mode Property Page

3) *Step Input (Figure 5-7)*

Formatted

The Step mode requires that a desired joint angle or position be entered, the Robot will then try to move to this position. The controller generates the joint PWM using a PI control algorithm.

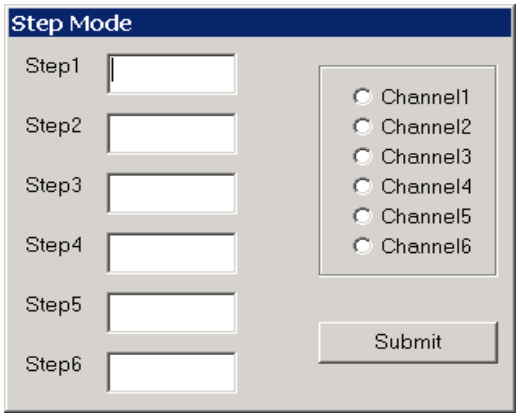

Figure 5-7 Step Mode Property Page

4) *Ramp Input (*Figure 5-8*)*

The ramp input mode is similar to the Step mode, a ramp is defined by setting a start position, a finish position, and the time required to complete the operation. A ramp is then generated between the two points for the robot to track. The start position can be set to the current joint position by clicking the button next to the start position edit box.

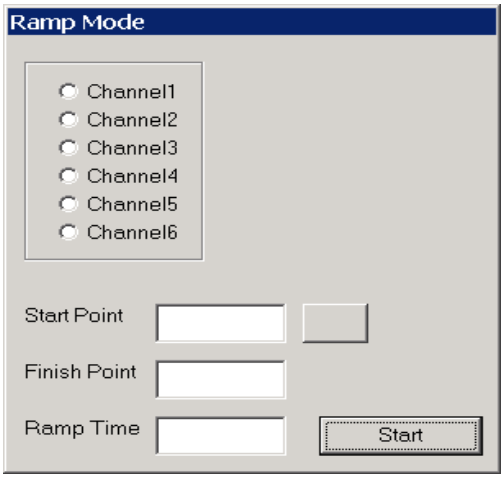

Figure 5-8 Ramp Mode Property Page

5) *Track Sinusoid Input (Figure 5-9)*

Formatted

The user enters values for amplitude, frequency and base value. A desired trajectory will then be created as follows:

$$DesiredPosition = A \times \sin(2 \times \pi \times f \times t) + BaseVal$$

The controller will then track this sinusoid.

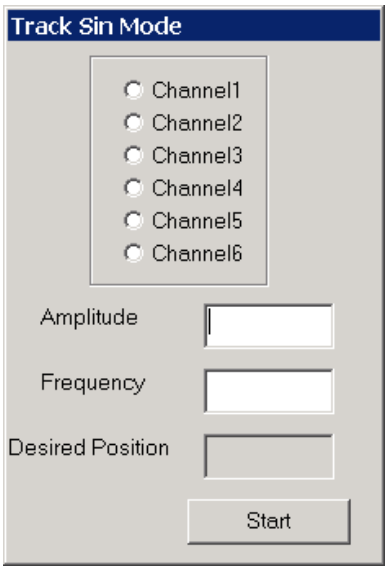

Figure 5-9 Track Sin Property Page

6) *Track User Defined Path*

A user defined path needs to be loaded into the program workspace. User defined paths are stored as text files, the text files are read into the program and a path is sent through to the controller. The controller tracks this path using the same methods as above. The creation of the path file is discussed later in section 5.2.1.

7) *Joy Stick Control*

As expected this mode allows control of the robot using the joystick, as the Joystick is a 2 degree of freedom joystick – and the robot has 6 degrees of freedom only 2 degrees of freedom can be exercised at a time. The choice of which degrees of freedom are activated requires the user to hold down the

appropriate keys on the keyboard (instructions are given in the program). Rather than manipulate the robot in joint space (i.e. change the joint variables), the joystick moves the robot in task space. This requires the solution of the inverse Jacobian matrix:

$$\tilde{\dot{q}} = J^{-1}\tilde{\dot{p}} \qquad \textbf{(5-1)}$$

where $\tilde{\dot{q}}$ is the matrix of joint velocities

$$\tilde{\dot{q}} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix}$$

and $\tilde{\dot{p}}$ is the matrix of end effector transformation velocities, this is made up of the velocities in the x, y, and z directions, and the RPY velocities for alpha, beta and gamma.

$$\tilde{\dot{p}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}$$

The output from the joystick is scaled such that it becomes a representation of the velocities described by the $\tilde{\dot{p}}$ matrix. The derived joint velocities from solving equation **(5-1)** are then linearly converted to PWM values. This linear conversion from the joint velocity to PWM value is not entirely accurate, as the relationship between the joint velocities and the PWM value for each joint is not linear, however this does provide a reasonable approximation. This approach is not ideal, and a better solution would be to use the tracking capabilities of the controller, which would require that a path be created “on the fly”. This approach however is quite complicated, and it has not been implemented in this application.

Other Functionality

The Main GUI also performs other tasks

1) The Robot can be stopped at any time by pressing the stop button on the main GUI. The start button must be pressed to restart its movement.
2) Plot a Graph showing current position, desired position, and PWM effort for each joint.
3) Write the joint data to a file. Selecting the print options dialog can change the data that is written to file.
4) Launch the Path Generator GUI.

Formatted: Bullets and Numbering

5.4.2 Path Generation GUI

This GUI is launched from the main GUI through the PathGenerator menu. As explained in chapter 6 a trajectory is made up of a number of segments. The GUI allows data to be entered for each segment. A new segment may be added to the trajectory through the use of the Segment menu. Selecting the appropriate segment number from the Segment dropdown box, and changing the data appropriately allows editing of the segments. The joint path for each of the joints is automatically plotted when the number of segments is greater than five. The path can be saved so that it can be loaded later by the path generator for editing, or it can be exported to a format that can be read by the path tracking option in the main GUI. The specifics of the path generating algorithms are discussed in chapter 6.

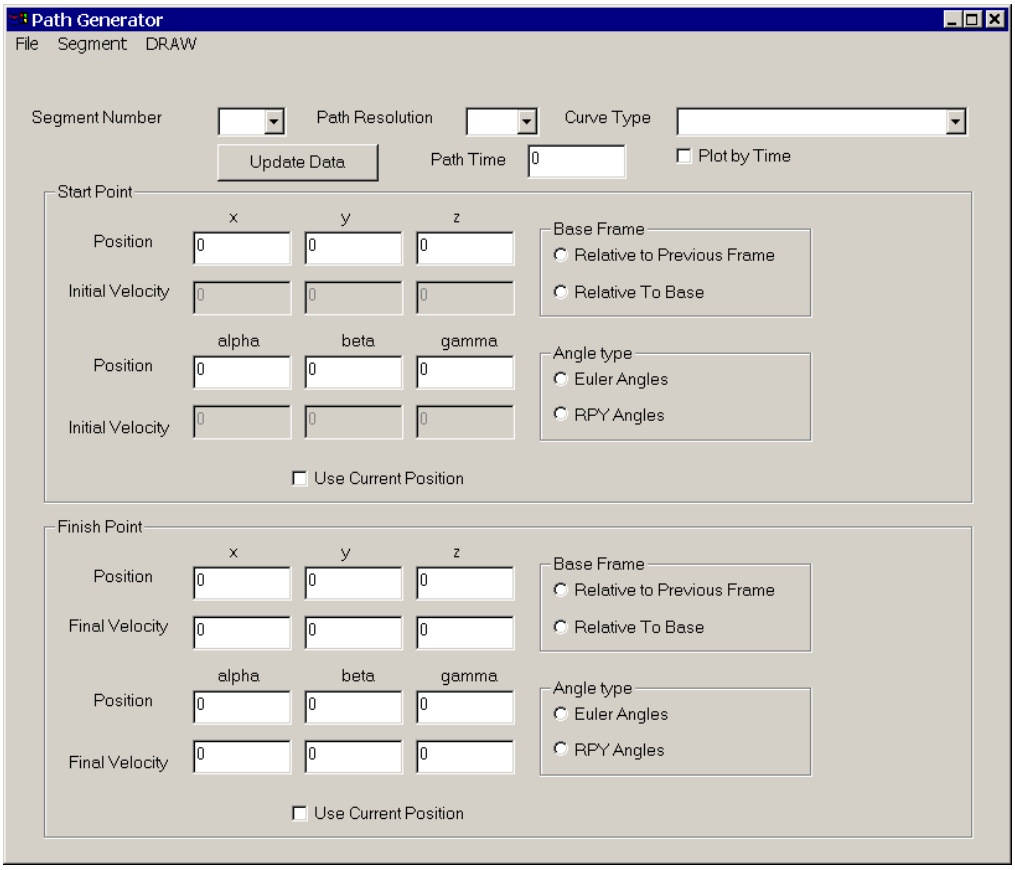

Figure 5-10 Path Generation GUI

Figure 5-11 shows the event map of the Path Generator program, the specifics of each of the functions mentioned in Figure 5-11 can be found in the code listings. Figure 6-1 also shows some of the working involved in generating the path.

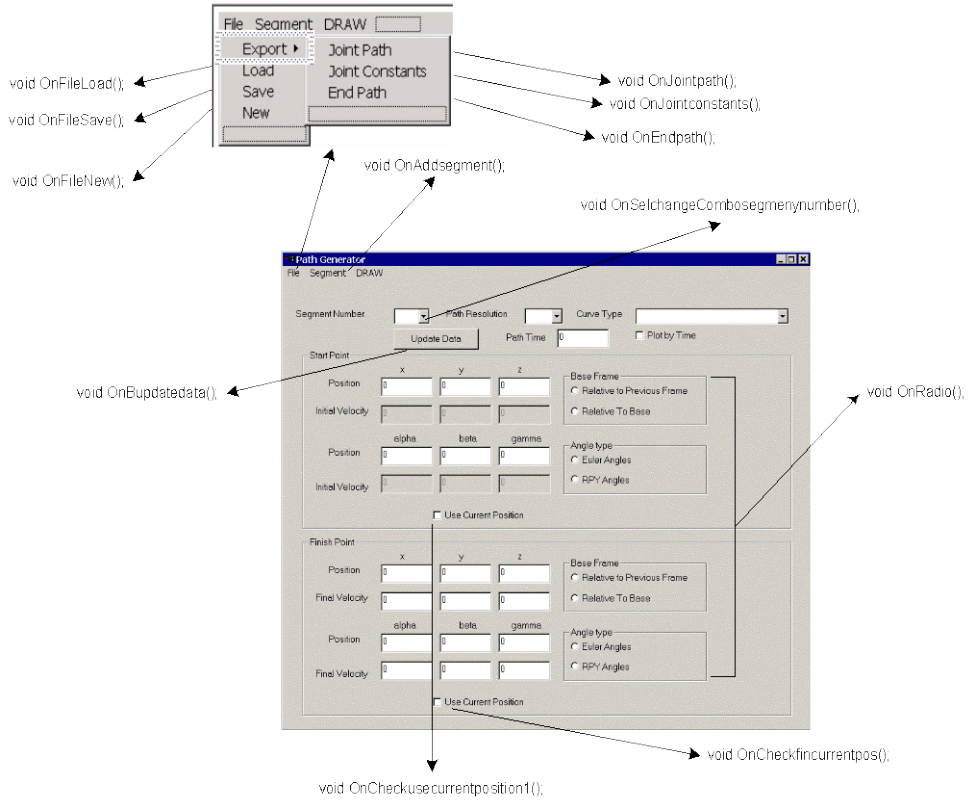

Figure 5-11 Path Generator Dialog Control Event Map

6 Trajectory Generation

The goal of trajectory planning is to generate a path for each of the manipulator joints to follow. This is done by defining an end effector path with a number of points (or knots) that the manipulator should pass through. These knots are chosen to define the path in critical areas and to ensure that the manipulator remains within the workspace. A cubic spline is used for spatial interpolation through the end effector knots. A time law is then applied to this spatial spline and the resulting spline is then sampled, with the inverse kinematics being solved to obtain a set of joint knots. The sampling resolution used should be greater in critical areas. Another spline is fitted to this set of joint knots to provide a set of spline coefficients that define the joint paths. These spline coefficients are directly used by the real time controller to determine the desired trajectory at the required time instant.

The following terms will be used in this section:

- *Path*: A path denotes the locus of points that the manipulator must follow; this may be defined in either joint space or workspace coordinates.
- *Trajectory*: A trajectory is a time dependant path.
- *Knot*: A point used to define the position of the manipulator; this may be defined in either joint space or workspace coordinates.
- *Segment:* Refers to a subsection of a path or trajectory. Thus for a spline a segment will be one of the single cubic polynomials between adjacent knots that define the spline.

6.1 Outline

A joint trajectory is required for each of the robot's 6 joints. The joint trajectory will be defined as an array of the following form:

Joint Trajectory = $\left[\tilde{B} \quad \tilde{W}\right]$

where $\tilde{B}$ is an array defining the constants of a spatial cubic spline

$$\tilde{\tilde{B}} = \begin{bmatrix} \tilde{\tilde{B}}_{1,1} & \tilde{\tilde{B}}_{1,2} & \tilde{\tilde{B}}_{1,3} & \tilde{\tilde{B}}_{1,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{B}}_{i,1} & \tilde{\tilde{B}}_{i,2} & \tilde{\tilde{B}}_{i,3} & \tilde{\tilde{B}}_{i,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{B}}_{m-1,1} & \tilde{\tilde{B}}_{m-1,2} & \tilde{\tilde{B}}_{m-1,3} & \tilde{\tilde{B}}_{m-1,4} \end{bmatrix},$$

$$\tilde{\tilde{B}}_{i,j} = \begin{bmatrix} B_{\theta_1,i,j} & B_{\theta_2,i,j} & B_{d_3,i,j} & B_{\theta_4,i,j} & B_{\theta_5,i,j} & B_{\theta_6,i,j} \end{bmatrix}$$

and $\tilde{\tilde{W}}$ is the time at the end of each of the m joint segments:

$$\tilde{\tilde{W}} = \begin{bmatrix} w_1 \\ \vdots \\ w_i \\ \vdots \\ w_{m-1} \end{bmatrix}$$

The joint position at any time t can be found from **(6-1)**.

$$\tilde{Q}_i(t) = \tilde{\tilde{B}}_{i,1} + \tilde{\tilde{B}}_{i,2}t + \tilde{\tilde{B}}_{i,3}t^2 + \tilde{\tilde{B}}_{i,4}t^3, 0 \le t \le (w_{i+1} - w_i),\ i = 1,\ldots,m-1 \qquad \textbf{(6-1)}$$

However it is neither practical nor desirable to design a manipulator trajectory using joint positions. The trajectory should be designed using a desired end effector trajectory, and then inverse kinematics used to solve for the joint trajectory. This is the approach that is used in this thesis. Figure 6-1 shows an overview of how a path is constructed.

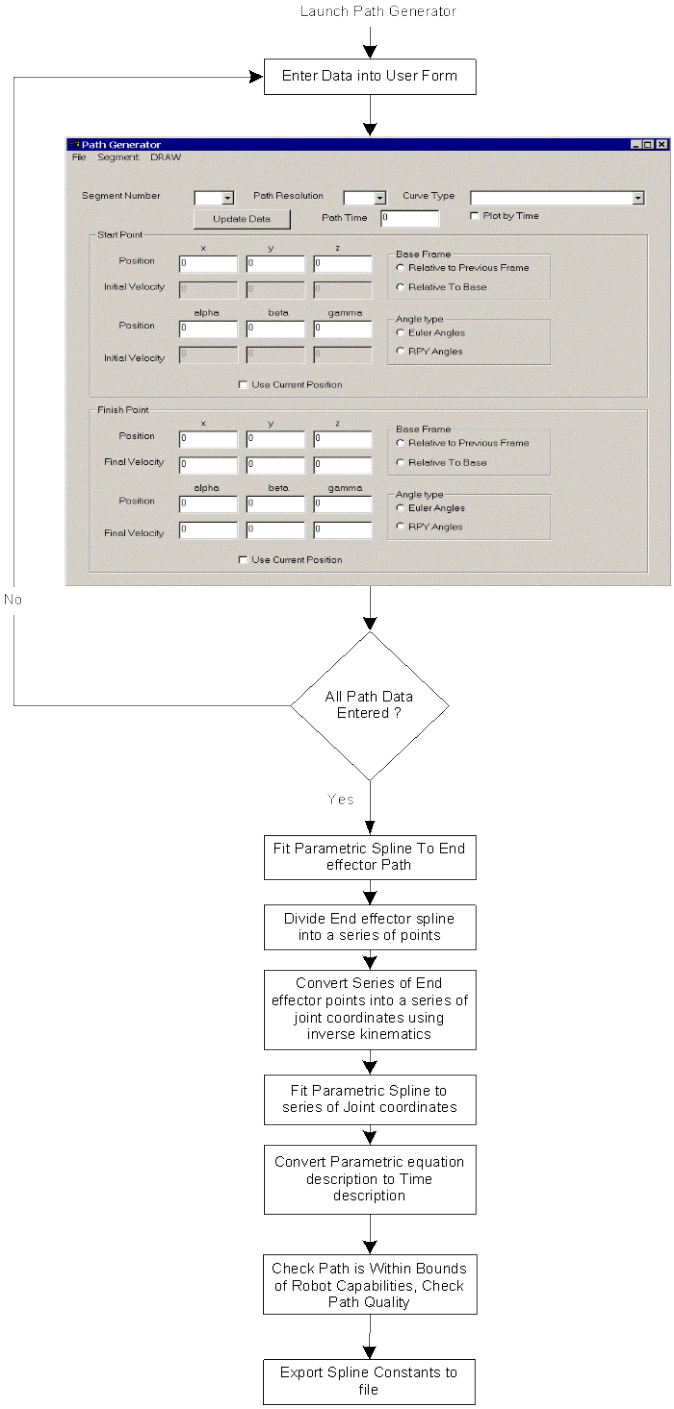

Figure 6-1 Overview of Path Generation

The following steps are a more detailed description of how a path is created:

6.1.1 Step 1 Define End Points:

Using the path generator GUI define a vector $\tilde{S}$ which describes the n end effector knots that the robot must pass through of the form:

$$\tilde{S} = \begin{bmatrix} \tilde{X}_1 \\ \vdots \\ \tilde{X}_k \\ \vdots \\ \tilde{X}_n \end{bmatrix}, \quad 1 \le k \le n,$$

where $\tilde{X}_k = [x_k, y_k, z_k, \alpha_k, \beta_k, \gamma_k]$ is the end effector coordinates at each knot.

Define another vector $\tilde{T}$ of the form:

$$\tilde{T} = \begin{bmatrix} t_1 \\ \vdots \\ t_k \\ \vdots \\ t_n \end{bmatrix}$$

which describes the time at which each knot should be reached.

The n points will generate n-1 segments.

6.1.2 Step 2 Generate Parametric Path Through End Points:

Parametric cubic spline interpolation through the knots in $\tilde{S}$ will yield a spatial description of the manipulator path of the form:

$$\tilde{S}(r) = \tilde{A} \begin{bmatrix} 1 \\ r \\ r^2 \\ r^3 \end{bmatrix}, 0 \le r \le 1$$

where $\tilde{S}(r)$ is of the same form as $\tilde{S}$, and $\tilde{A}$ describes the constants of a cubic spline:

$$\tilde{\tilde{A}} = \begin{bmatrix} \tilde{\tilde{A}}_{1,1} & \tilde{\tilde{A}}_{1,2} & \tilde{\tilde{A}}_{1,3} & \tilde{\tilde{A}}_{1,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{A}}_{k,1} & \tilde{\tilde{A}}_{k,2} & \tilde{\tilde{A}}_{k,3} & \tilde{\tilde{A}}_{k,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{A}}_{n-1,1} & \tilde{\tilde{A}}_{n-1,2} & \tilde{\tilde{A}}_{n-1,3} & \tilde{\tilde{A}}_{n-1,4} \end{bmatrix},$$

and

$$\tilde{\tilde{A}}_{i,j} = [a_{x,i,j}, a_{y,i,j}, a_{z,i,j}, a_{\alpha,i,j}, a_{\beta,i,j}, a_{\gamma,i,j}]$$

The value of **X** at any point on segment *k* can be found using **(6-2)**.

$$\tilde{\tilde{S}}_k(r) = A_{k,1} + A_{k,2}r + A_{k,3}r^2 + A_{k,4}r^3,\ 0 \le r < 1 \qquad \textbf{(6-2)}$$

It should also be noted that:

$$\tilde{\tilde{S}}_k(1) = \tilde{\tilde{S}}_{k+1}(0) = \tilde{\tilde{X}}_{k+1}$$

However spline interpolation requires an initial and final gradient, but as this is not available, linear interpolation is applied to the first and last segments, providing a starting and finishing gradient for spline interpolation on the range $2 \le k \le n-2$.

Linear interpolation on the first and last segments yields the following constants in the spline constants matrix:

For the first segment (k=1):

$$\tilde{\tilde{A}}_{1,1} = \tilde{\tilde{X}}_1$$

$$\tilde{\tilde{A}}_{1,2} = (\tilde{\tilde{X}}_2 - \tilde{\tilde{X}}_1)$$

$$\tilde{\tilde{A}}_{1,3} = \tilde{\tilde{A}}_{1,4} = 0$$

For the last segment (k=n-1):

$$\tilde{\tilde{A}}_{n-1,1} = \tilde{\tilde{X}}_{n-1}$$

$$\tilde{\tilde{A}}_{n-1,2} = (\tilde{\tilde{X}}_n - \tilde{\tilde{X}}_{n-1})$$

$$\tilde{\tilde{A}}_{n-1,3} = \tilde{\tilde{A}}_{n-1,4} = 0$$

Spline interpolation on the range $2 \le k \le n-2$ using the methods described in appendix D is combined with the results above to yield the following matrix of spline constants:

$$\tilde{A} = \begin{bmatrix} \tilde{A}_{1,1} & \tilde{A}_{1,2} & 0 & 0 \\ \tilde{A}_{2,1} & A_{2,2} & \tilde{A}_{2,3} & \tilde{A}_{2,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{A}_{k,1} & \tilde{A}_{k,2} & \tilde{A}_{k,3} & \tilde{A}_{k,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{A}_{n-2,1} & \tilde{A}_{n-2,2} & \tilde{A}_{n-2,3} & \tilde{A}_{n-2,4} \\ \tilde{A}_{n-1,1} & \tilde{A}_{n-1,2} & 0 & 0 \end{bmatrix},$$

where:

$$\tilde{A}_{i,j} = [a_{x,i,j}, a_{y,i,j}, a_{z,i,j}, a_{\alpha,i,j}, a_{\beta,i,j}, a_{\gamma,i,j}]$$

The value of the spline can be evaluated using **(6-2)**.

6.1.3 Step 3 Convert Parametric Path to Time Dependant Path:

The interpolated end effector path must then be converted into a trajectory (time dependent path). This is achieved by creating a time law for each of the parametric equations, and means that each segment will have a parametric equation describing the path $\tilde{S}(r)$ of the form:

$$\tilde{S}(r) = diag\left(\tilde{A}\tilde{R}\right) = \begin{bmatrix} \tilde{A}_1\tilde{R}_1 \\ \vdots \\ \tilde{A}_k\tilde{R}_k \\ \vdots \\ \tilde{A}_{n-1}\tilde{R}_{n-1} \end{bmatrix} \qquad \textbf{(6-3)}$$

Where the parametric quantity is defined as:

$$\tilde{R} = \begin{bmatrix} \tilde{R}_1 & \cdots & \tilde{R}_k & \cdots & \tilde{R}_{n-1} \end{bmatrix}, \qquad \textbf{(6-4)}$$

where

$$\tilde{R}_i = \begin{bmatrix} \tilde{r}_i^{\,0} \\ \tilde{r}_i^{\,1} \\ \tilde{r}_i^{\,2} \\ \tilde{r}_i^{\,3} \end{bmatrix}$$

and

$$\tilde{r}_i^{\,e} = \left[r_{i,\theta_1}^e, r_{i,\theta_2}^e, r_{i,d_3}^e, r_{i,\theta_4}^e, r_{i,\theta_5}^e, r_{i,\theta_6}^e\right],\ e = 0,1,2\ or\ 3$$

with each element $\tilde{r}_k$ being a time dependent function of the form **(6-5)**.

$$\tilde{r}_k = \tilde{C}_{k,1} + \tilde{C}_{k,2}(t - T_k) + \tilde{C}_{k,3}(t - T_k)^2 + \tilde{C}_{k,4}(t - T_k)^3,\ T_k \le t < T_{k+1} \qquad \textbf{(6-5)}$$

The constant matrix $\tilde{C}$ represents the constants that make the cubic polynomial time law. With $\tilde{C}$ defined as:

$$\tilde{C} = \begin{bmatrix} \tilde{C}_1 \\ \vdots \\ \tilde{C}_k \\ \vdots \\ \tilde{C}_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & \tilde{V}_1 & \frac{3-(2\tilde{V}_1+\tilde{V}_2)\tau_1}{\tau_1^2} & \frac{\tau_1(\tilde{V}_1+\tilde{V}_2)-2}{\tau_1^3} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \tilde{V}_k & \frac{3-(2\tilde{V}_k+\tilde{V}_{k+1})\tau_k}{\tau_k^2} & \frac{\tau_k(\tilde{V}_k+\tilde{V}_{k+1})-2}{\tau_k^3} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \tilde{V}_{n-1} & \frac{3-(2\tilde{V}_{n-1}+\tilde{V}_n)\tau_{n-1}}{\tau_{n-1}^2} & \frac{\tau_{n-1}(\tilde{V}_{n-1}+\tilde{V}_n)-2}{\tau_{n-1}^3} \end{bmatrix} \qquad \textbf{(6-6)}$$

where

$$\tilde{\tau} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_k \\ \vdots \\ \tau_{n-1} \end{bmatrix} = \begin{bmatrix} T_2 - T_1 \\ \vdots \\ T_{k+1} - T_k \\ \vdots \\ T_n - T_{n-1} \end{bmatrix}$$

and $\tilde{V}$ is the user defined velocity constants that control the shape of the time law (see Figure 6-2):

$$\tilde{V} = \begin{bmatrix} \tilde{V}_1 \\ \vdots \\ \tilde{V}_k \\ \vdots \\ \tilde{V}_n \end{bmatrix},$$

where

$$\tilde{V}_k = [V_{x,k}, V_{y,k}, V_{z,k}, V_{\alpha,k}, V_{\beta,k}, V_{\gamma,k}]$$

The derivation of the constants in **(6-6)** follows:

Given the equation **(6-5)** which gives the time law for the parameter r_k on the range $T_k \le t < T_{k+1}$:

$$\tilde{r}_k = \tilde{C}_{k,1} + \tilde{C}_{k,2}(t - T_k) + \tilde{C}_{k,3}(t - T_k)^2 + \tilde{C}_{k,4}(t - T_k)^3,$$

the derivatives are

$$\dot{\tilde{r}}_k(t) = \tilde{C}_{k,2} + 2\tilde{C}_{k,3}(t - T_k) + 3\tilde{C}_{k,4}(t - T_k)^2 \quad \textbf{(6-7)}$$

$$\ddot{\tilde{r}}_k(t) = 2\tilde{C}_{k,3} + 6\tilde{C}_{k,4}(t - T_k) \quad \textbf{(6-8)}$$

with the initial conditions defined as:

$\tilde{r}_k(T_k) = 0$
$\tilde{r}_k(T_{k+1}) = 1$
$\dot{\tilde{r}}_k(T_k) = \tilde{V}_k$
$\dot{\tilde{r}}_k(T_{k+1}) = \tilde{V}_{k+1}$

solving equations **(6-5)**,**(6-7)** and **(6-8)** with initial conditions above yields the following cubic constants:

$$\tilde{C}_{k,1} = 0 \quad \textbf{(6-9)}$$

$$\tilde{C}_{k,2} = V_k \quad \textbf{(6-10)}$$

$$\tilde{C}_{k,3} = \frac{3 - (2\tilde{V}_k + \tilde{V}_{k+1})\tau_k}{\tau_k^2} \quad \textbf{(6-11)}$$

$$\tilde{C}_{k,4} = \frac{\tau_k(\tilde{V}_k + \tilde{V}_{k+1}) - 2}{\tau_k^3} \quad \textbf{(6-12)}$$

A graphical view of time law for the parameter in the parametric equations can be seen in Figure 6-2.

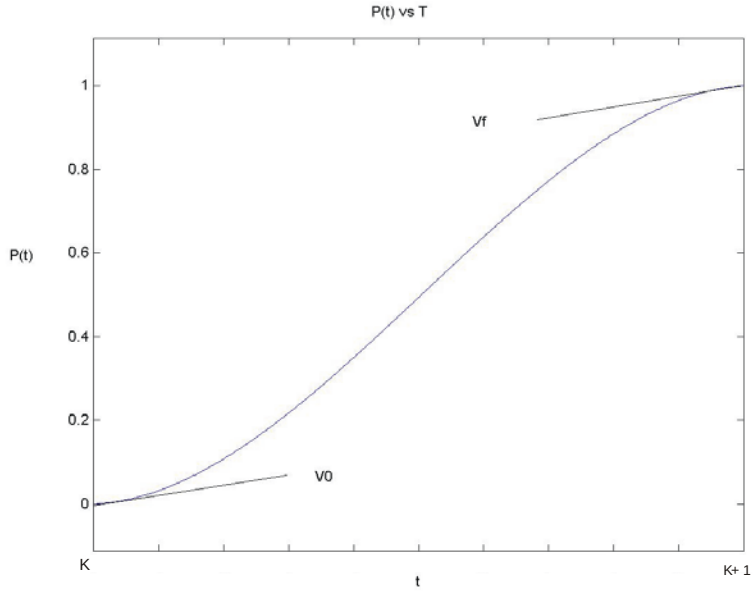

Figure 6-2 Time Law for Parametric Path

Combining the Parametric Path with the corresponding time law yields a description of the end effector trajectory as shown in equation **(6-13)**:

$$\tilde{S}(t) = diag\left(\tilde{A} \times \begin{bmatrix} 1 \\ \tilde{C}t \\ (\tilde{C}t)^2 \\ (\tilde{C}t)^3 \end{bmatrix} \right) \qquad \textbf{(6-13)}$$

6.1.4 Step 4: Create Joint Space Trajectory

With the end effector trajectory defined a joint space trajectory must now be created. Take each segment $\tilde{S}(t)_i$ and evaluate it at j_i equally spaced time intervals Δ_i so that range $T_i + j_i\Delta_i = T_{i+1}$.

This yields the array $\tilde{P}_i$ for each segment:

$$\tilde{\tilde{P}}_i = \begin{bmatrix} S(T_i) \\ \vdots \\ S(T_i + h\Delta_i) \\ \vdots \\ S(T_i + (j-1)\Delta_i) \end{bmatrix} = \begin{bmatrix} X_{i,1} \\ \vdots \\ X_{i,h} \\ \vdots \\ X_{i,j} \end{bmatrix},\ 0 \le h \le j_i - 1$$

The array for the whole trajectory $\tilde{\tilde{P}}$ will then be of the form:

$$\tilde{\tilde{P}} = \begin{bmatrix} \tilde{\tilde{P}}_1 \\ \vdots \\ \tilde{\tilde{P}}_i \\ \vdots \\ \tilde{\tilde{P}}_n \end{bmatrix}$$

or

$$\tilde{\tilde{P}} = \begin{bmatrix} \tilde{\tilde{X}}_1 \\ \vdots \\ \tilde{\tilde{X}}_i \\ \vdots \\ \tilde{\tilde{X}}_m \end{bmatrix},\ 1 \le i \le m,\ m = \left(\sum_{i=1}^{n} j_i \right) + 1$$

The corresponding time vector $\tilde{\tilde{W}}$ is defined as:

$$\tilde{\tilde{W}} = \begin{bmatrix} \tilde{\tilde{W}}_1 \\ \vdots \\ \tilde{\tilde{W}}_i \\ \vdots \\ \tilde{\tilde{W}}_n \end{bmatrix},$$

where

$$\tilde{\tilde{W}}_i = \begin{bmatrix} T_i \\ \vdots \\ T_i + h\Delta_i \\ \vdots \\ T_i + (j-1)\Delta_i \end{bmatrix},\ 0 \le h \le j_i - 1$$

or

$$\tilde{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_i \\ \vdots \\ w_m \end{bmatrix}, 1 \le i \le m,\ m = \left(\sum_{i=1}^{n} j_i \right) + 1$$

Thus the new end effector trajectory $\tilde{U}$ is:

$$\tilde{U} = \begin{bmatrix} \tilde{P} & \tilde{W} \end{bmatrix}$$

The Inverse Kinematics is calculated from each row entry, to create the corresponding joint space trajectory. To calculate the inverse kinematics the end effector transformation matrix must be created, this can be done using **(6-14)**, which is based on the RPY method of representing the angles.

$$T^{06} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha c\gamma & P_x \\ s\alpha c\beta & s\alpha s\beta s\gamma - c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & P_y \\ -s\beta & c\beta s\gamma & c\beta c\gamma & P_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(6-14)}$$

Once the inverse kinematics are solved, the joint trajectory **O** will be represented by:

$$\tilde{O} = \begin{bmatrix} \tilde{Q} & \tilde{W} \end{bmatrix}$$

where $\tilde{Q}$ is the joint position array:

$$\tilde{Q} = \begin{bmatrix} \tilde{Q}_1 \\ \vdots \\ \tilde{Q}_i \\ \vdots \\ \tilde{Q}_m \end{bmatrix},$$

where,

$$\tilde{Q}_i = \begin{bmatrix} \theta_{i,1}, \theta_{i,2}, d_{i,3}, \theta_4, \theta_5, \theta_6 \end{bmatrix}$$

A spline must then be fitted to the joint trajectories using the methods in appendix D, with the initial and final velocities set to zero. The resulting spline constants $\tilde{B}$ are defined as:

$$\tilde{\tilde{B}} = \begin{bmatrix} \tilde{\tilde{B}}_{1,1} & \tilde{\tilde{B}}_{1,2} & \tilde{\tilde{B}}_{1,3} & \tilde{\tilde{B}}_{1,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{B}}_{i,1} & \tilde{\tilde{B}}_{i,2} & \tilde{\tilde{B}}_{i,3} & \tilde{\tilde{B}}_{i,4} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\tilde{B}}_{m-1,1} & \tilde{\tilde{B}}_{m-1,2} & \tilde{\tilde{B}}_{m-1,3} & \tilde{\tilde{B}}_{m-1,4} \end{bmatrix}$$

where

$$\tilde{Q} = \tilde{\tilde{B}} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Thus the joint position $\tilde{Q}$ can be found at any time t using **(6-15)**.

$$\tilde{\tilde{Q}}_k = \tilde{\tilde{B}}_{k,1} + \tilde{\tilde{B}}_{k,2} t + \tilde{\tilde{B}}_{k,3} t^2 + \tilde{\tilde{B}}_{k,4} t^3, \quad W_k \le t < W_{k+1}, \; 1 \le k \le m-1 \qquad \textbf{(6-15)}$$

6.1.5 Sample Trajectory

Figures 6-3 and 6-4 show an example of a trajectory. Figure 6-3 shows the end effector trajectory, and Figure 6-4 the corresponding joint trajectories.

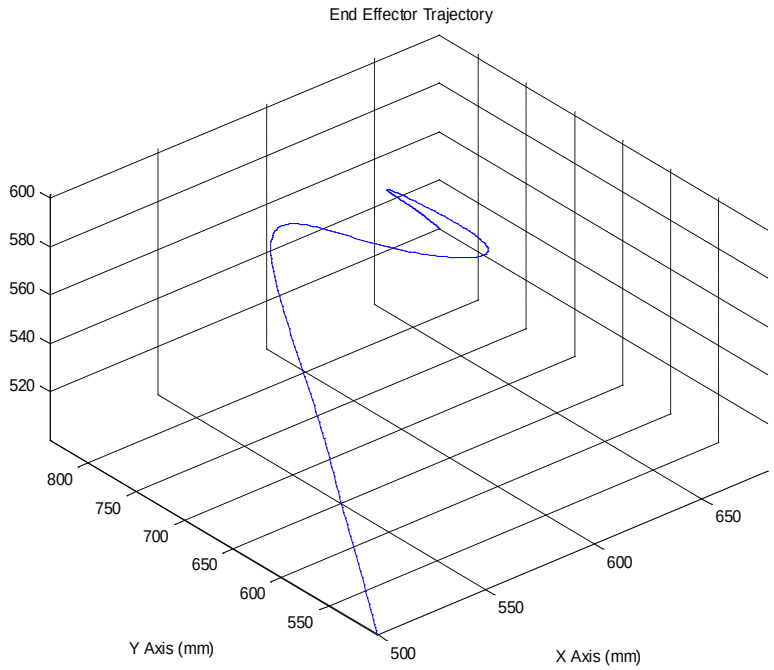

Figure 6-3 End Effector Trajectory

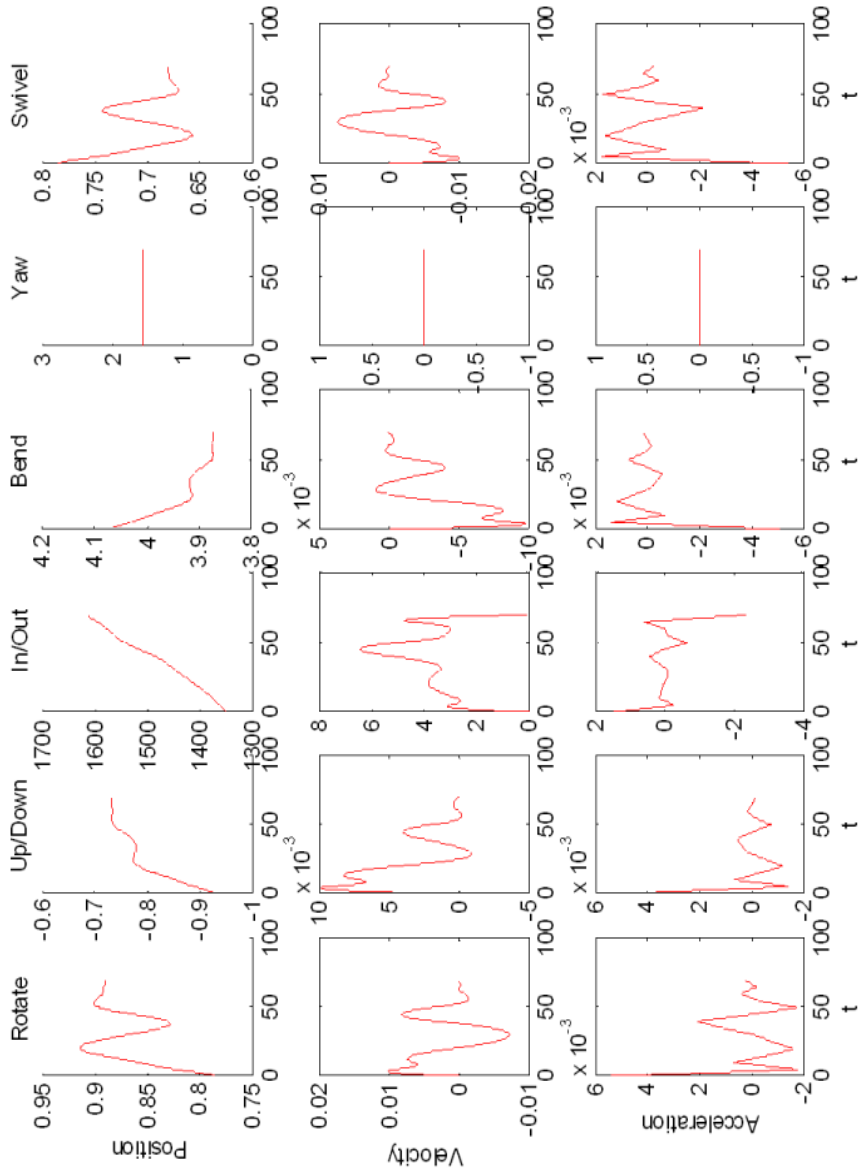

Figure 6-4 Joint Trajectory

6.2 Check Maximum Joint Parameters

The path needs to be checked to ensure that it does not exceed the limitations of the robot. Thus for each segment the position, velocity, acceleration are checked against the maximum allowable value for each joint.

6.2.1 Joint Position.

Given the joint segment trajectory **O**:

$$\tilde{O} = \begin{bmatrix} \tilde{Q} & \tilde{W} \end{bmatrix}$$

which has m-1 segments, with the position on any segment at a time t defined as:

$$\tilde{Q}_k = \tilde{B}_{k,1} + \tilde{B}_{k,2}t + \tilde{B}_{k,3}t^2 + \tilde{B}_{k,4}t^3, \quad W_k \le t < W_{k+1}, \; 1 \le k \le m-1$$

and

$$\dot{\tilde{Q}}_k = \tilde{B}_{k,2} + 2\tilde{B}_{k,3}t + 3\tilde{B}_{k,4}t^2 \qquad \textbf{(6-16)}$$

The maximum value of $\tilde{Q}$ is found at the time t when $\tilde{Q}$ is maximised, or at the time t when $\dot{\tilde{Q}}_k = 0$. Solving **(6-16)** for $\dot{\tilde{Q}}_k = 0$ yields the following value for t:

$$t_{k,i} = \frac{-B_{k,3,i} \pm \sqrt{B_{k,3,i}^2 - 3B_{k,4,i}B_{k,2,i}}}{3B_{k,4,i}}, \; 1 \le i \le 6 \qquad \textbf{(6-17)}$$

Where

$$\tilde{t}_k = \begin{bmatrix} t_{k,\theta_1} & t_{k,\theta_2} & t_{k,d_3} & t_{k,\theta_4} & t_{k,\theta_5} & t_{k,\theta_6} \end{bmatrix}$$

The maximum/minimum joint value on any segment k could then be found by substituting the solutions from **(6-17)** into **(6-18)**.

$$Q_{k,p} = B_{k,1,p} + B_{k,2,p}t_{k,p} + B_{k,3,p}t_{k,p}^{\;2} + B_{k,4,p}t_{k,p}^{\;3}, \quad 1 \le p \le 6, \; 1 \le k \le m-1 \qquad \textbf{(6-18)}$$

If the joint value $Q_{k,p}$ is outside the physical limits of the joint then the path should be revised.

6.2.2 Joint Velocity.

Differentiating the joint segment trajectory yields the joint velocity throughout the segment k gives **(6-19)** and **(6-20)**.

$$\dot{\tilde{Q}}_k = \tilde{B}_{k,2} + 2\tilde{B}_{k,3}t + 3\tilde{B}_{k,4}t^2 \qquad \textbf{(6-19)}$$

$$\ddot{\tilde{Q}}_k = 2\tilde{B}_{k,3} + 6\tilde{B}_{k,4}t \qquad \textbf{(6-20)}$$

The maximum value of $\ddot{\tilde{Q}}$ is found at the time t when $\ddot{\tilde{Q}}$ is maximised, or at the time t when $\ddot{\tilde{Q}}_k = 0$. Solving **(6-20)** for $\ddot{\tilde{Q}}_k = 0$ yields the following value for t:

$$\tilde{t}_k = -\frac{\tilde{B}_{k,3}}{3\tilde{B}_{k,4}}\ ,\ \tilde{t}_k = \begin{bmatrix} t_{k,\theta_1} & t_{k,\theta_2} & t_{k,d_3} & t_{k,\theta_4} & t_{k,\theta_5} & t_{k,\theta_6} \end{bmatrix} \qquad \textbf{(6-21)}$$

The maximum/minimum joint velocity on any segment k could then be found by substituting the solutions from **(6-21)** into **(6-23)**.

$$\dot{Q}_{k,p} = B_{k,2,p} + 2B_{k,3,p}t_{k,p} + 3B_{k,4,p}t_{k,p}{}^2,\ 1 \le p \le 6,\ 1 \le k \le m-1 \qquad \textbf{(6-22)}$$

If the joint velocity $\dot{Q}_{k,p}$ is outside the physical limits of the joint then the path should be revised.

6.2.3 Joint Acceleration.

Because the joint acceleration is a linear function over any segment k, the maximum acceleration must be either at the start of the segment or the end of the segment, therefore:

$\ddot{\tilde{Q}}_{k,\max} = 2\tilde{B}_{k,3}$, or $\ddot{\tilde{Q}}_{k,\max} = 2\tilde{B}_{k+1,3}$

If the joint acceleration $\ddot{Q}_{k,\max}$ is outside the physical limits of the joint then the path should be revised.

$$\ddot{y}_j(t)_{\max} = 2a_{2,j} \qquad \textbf{(6-23)}$$

6.3 Path Deviation

The path deviation is a measure of how much the derived joint paths deviate from the originally specified end effector path. If the path deviation is too great then Step 4 should be repeated, but with a smaller time slice, i.e. the end effector trajectory will be described by a greater number of points, inherently increasing the accuracy of the derived joint paths.

Given an end effector path segment defined as:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} a_{0x} & a_{1x} & a_{2x} & a_{3x} \\ a_{0y} & a_{1y} & a_{2y} & a_{3y} \\ a_{0z} & a_{1z} & a_{2z} & a_{3z} \end{bmatrix} \begin{bmatrix} 1 \\ k \\ k^2 \\ k^3 \end{bmatrix} \qquad \text{for} \quad 0 \le k \le 1$$

and joint path over the same segment as defined as

$$P_i = a_{0,i} + a_{1,i}k + a_{2,i}k^2 + a_{3,i}k^3 \qquad \text{for } 1 \le \text{i} \le 6$$

For any joint configuration the end effector position can be found using the forward kinematics. The forward kinematics used to derive the end effector position Fp will be expressed as follows:

$$\hat{P}_x = Fp_x(\theta_1, \theta_2, d_3, \theta_4, \theta_5, \theta_6)$$
$$\hat{P}_y = Fp_y(\theta_1, \theta_2, d_3, \theta_4, \theta_5, \theta_6)$$
$$\hat{P}_z = Fp_z(\theta_1, \theta_2, d_3, \theta_4, \theta_5, \theta_6)$$

6.3.1 Analytical Solution of Path deviation:

If the error is defined as the square of the difference between the desired trajectory and the actual trajectory. The error will then be described by **(6-24)**.

$$e = qq^T \qquad \textbf{(6-24)}$$

where

$$q = \begin{bmatrix} \hat{P}_x(k) \\ \hat{P}_y(k) \\ \hat{P}_z(k) \end{bmatrix} - \begin{bmatrix} P_x(k) \\ P_y(k) \\ P_z(k) \end{bmatrix} = \begin{bmatrix} Fp_x(k) - P_x(k) \\ Fp_y(k) - P_y(k) \\ Fp_z(k) - P_z(k) \end{bmatrix} \qquad 0 < k < 1.$$

$\hat{P}_i(k)$ is the estimate value of the path,

$Fp_i(k)$ is a function describing the forward kinematics of the joint variables over the range of the parametric variable k, and

$P_i(k)$ is the original end effector spline that the joint variables are base on i.e. a simple cubic polynomial.

The value of k that causes the maximum error over the range of $0 < k < 1$ is found by setting:

$$\frac{de}{dk} = 0$$

the maximum error for this segment can then be found by substituting this value of k into **(6-24)**.

solve $\frac{de}{dk}$ as follows:

$$\frac{de}{dk} = q\frac{d(q^T)}{dk} + q^T\frac{d(q)}{dk}$$

if define error Jacobian E as:

$$E = \begin{bmatrix} \frac{d}{dk}\left(Fp_x(k) - P_x(k)\right) \\ \frac{d}{dk}\left(Fp_y(k) - P_y(k)\right) \\ \frac{d}{dk}\left(Fp_z(k) - P_z(k)\right) \end{bmatrix}$$

then

$$\frac{de}{dk} = qE^T + q^T E = 0 \qquad \textbf{(6-25)}$$

Equation **(6-25)** yields a complicated set of equations in k, which are be difficult to solve analytically, and which yields multiple solutions. Hence it is impractical to find the maximum path deviation analytically. The approach laid out in section 6.3.2 is far

less complicated, and while it will not calculate an exact measure of the path deviation it does calculate an upper bound (c.f. Figure 6-5).

6.3.2 Approximate measure of Path Quality

Given the Robot Jacobian Matrix the following relation can be shown:

$$\tilde{v} = \tilde{J}(\tilde{q})\dot{\tilde{q}}$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \tilde{J} \times \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix} \qquad \textbf{(6-26)}$$

if we find **v** for the start and endpoint of the segment k ($\mathbf{v}_k$, $\mathbf{v}_{k+1}$), then the following geometric interpretation of the path can be made:

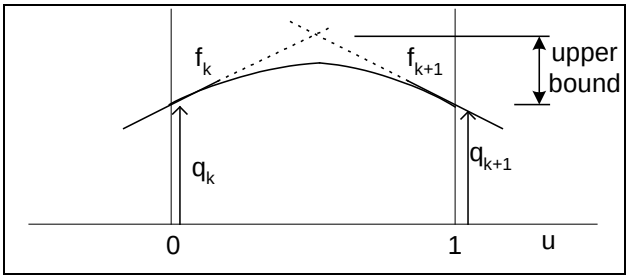

Figure 6-5 Geometric Interpretation of Path Deviation Measure

Where:

$$f_k(\tilde{u}) = \tilde{q}_k + \tilde{v}_k\tilde{u}$$

$$f_{k+1}(\tilde{u}) = \tilde{q}_{k+1} + \tilde{v}_{k+1}(1-\tilde{u})$$

where

$$\tilde{u} = [u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6],\ 0 \le u_i \le 1$$

A quantitative measure of the path deviation can be described by f_{max}, which is the point of intersection of the functions $f_k(\tilde{u})$ and $f_{k+1}(\tilde{u})$. At the intersection point:

$$f_k(\tilde{u}) - f_{k+1}(\tilde{u}) = 0$$

Therefore:

$$f_k(\tilde{u}) - f_{k+1}(\tilde{u}) = \tilde{q}_1 + \tilde{v}_k\tilde{u} - \tilde{q}_2 - \tilde{v}_{k+1} + \tilde{v}_{k+1}\tilde{u} = 0$$

and

$$\tilde{u} = \frac{(\tilde{v}_1 + \tilde{q}_2 - \tilde{q}_1)}{(\tilde{v}_0 + \tilde{v}_1)}$$

$$\tilde{f}_{\max} = \tilde{q}_1 + \tilde{v}_0\left(\frac{(\tilde{v}_1 + \tilde{q}_2 - \tilde{q}_1)}{(\tilde{v}_0 + \tilde{v}_1)}\right),$$

where

$$\tilde{f}_{\max} = \left[f_{\max_{k,1}}, f_{\max_{k,2}}, f_{\max_{k,3}}, f_{\max_{k,4}}, f_{\max_{k,5}}, f_{\max_{k,6}}\right]$$

$$v_{0,i} = a_{1,i}$$

$$v_{1,i} = a_{1,i+1}$$

A segment quality for the k[th] segment Q_k may then be measured by taking the product of $f_{\max k}$ for each of the 6 parameters describing the end effector position as shown in **(6-27)**.

$$Q_k = (f_{\max_{k,1}}(P_x))^2 + (f_{\max_{k,2}}(P_y))^2 + (f_{\max_{k,3}}(P_z))^2 + \Psi\left[(f_{\max_{k,4}}(\alpha))^2 + (f_{\max_{k,5}}(\beta))^2 + (f_{\max_{k,6}}(\gamma))^2\right] \quad \textbf{(6-27)}$$

Where ψ is a weighting factor to compensate for α, β, γ being angles, compared to the lengths Px, Py, Pz.

6.4 Create Virtual Robot Workspace

A virtual robot workspace can be created using SolidWorks which allows the robot, and any obstacles to be modelled. This virtual workspace can be used to develop and test Robot paths offline. SolidWorks has a VBA programming API, which allows a Visual Basic program to drive the SolidWorks model. The VBA program can in turn be used with C++ programs through the use of DLL's (dynamic link libraries). The VBA scripts can be used to measure or set the 6 joint variables. The model can then be used to follow a joint path created using the path generating software discussed in section 5.4.2. This can be useful for collision checking, since collisions can be

detected either visually, or by the use of collision detection tools built into SolidWorks. The model can also be used to create paths, as the model can be easily moved around manually (i.e. using the mouse, or defining joint/end effector variables). A path can then be generated by using the methods discussed previously, but instead of measuring the joint angles from the robot or from desired end effector position they can be measured from the model.

Figure 6-6 SolidWorks Model of Unimate 2000B

This approach can also be used to simulate multiple moving objects in a single workspace. The advantage of this is that possibility of collisions could be detected in the development stages rather than at run time.

7 Discussion

7.1 Hardware difficulties

Hardware difficulties noted in previous work [Yang *et al.* 2000] were eliminated with proper use of all bits of information from encoders, and this has seen the maximum encoder reading rate rise from 27Hz to 1kHz. Erroneous readings from the Yaw encoder were corrected using a software fix (see Appendix G for details). The staggered effect of reading each encoder every 6th interrupt is not an ideal situation, for more accurate control it would be necessary to read all 6 encoders at once. This could be achieved by fitting another FPGA to the robot near the encoders and reading all of the encoders continuously. The encoder positions can then be sampled at one instant and then read from the FPGA sequentially using the existing hardware and software.

7.2 Controller

An alternate approach to the controller design would be to use a model based controller, however this has two major difficulties:

1. the lack of an appropriate model of the system, and
2. any model of the system will be highly non-linear and heavily coupled.

Formatted: Bullets and Numbering

These difficulties could be overcome by the development of an adaptive controller *[Sciavicco and Siciliano]*. The important aspect of the adaptive controller is that an accurate model of all the system dynamics is available, however the parameters need not be accurate.

Suppose the exact model of the system is of the form:

$$\begin{aligned} u &= B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) \\ &= Y(q,\dot{q},\ddot{q})\pi \end{aligned} \qquad \textbf{(7-1)}$$

where π is a vector of constant parameters.

A suitable model based controller could then be of the form of **(7-2)**.

$$u = \hat{B}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{F}\dot{q}_r + \hat{g} + K_D\sigma$$
$$= Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{\pi} + K_D\sigma \qquad \textbf{(7-2)}$$

where:

$\hat{B},\hat{C},\hat{F},\hat{g},\tilde{\pi}$ are estimates of the parameters B,C,F,g,π respectively.

$\dot{q}_r = \dot{q}_d + \Lambda\tilde{q},\ \ddot{q}_r = \ddot{q}_d + \Lambda\dot{\tilde{q}}$

$\sigma = \dot{q}_r - \dot{q} = \dot{\tilde{q}} + \Lambda\tilde{q}$

and Λ is a positive definite matrix that allows expressing the non-linear compensation and coupling terms as a function of the desired velocity and acceleration.

By satisfying the Lyapunov stability criteria an adaptive parameter estimation law of the form shown in **(7-3)** can be shown to be bounded, with asymptotical convergence of $\sigma \to 0$ and $\tilde{q} \to 0$.

$$\dot{\hat{\pi}} = K_\pi^{-1} Y^T(q,\dot{q},\dot{q}_r,\ddot{q}_r)\sigma \qquad \textbf{(7-3)}$$

where:

K_π is a symmetric positive definite matrix

Combining equations **(7-1)**, **(7-2)**, and **(7-3)** in a diagrammatic form (Figure 7-1) allows an easier interpretation of the system.

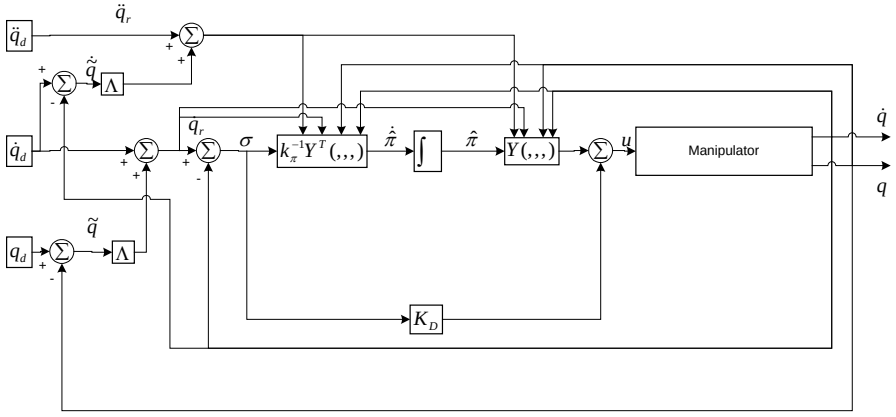

Figure 7-1 Adaptive Controller

7.3 Integration with Matlab

Matlab is useful for the design and analysis of control systems, however it is not as useful for the deployment and implementation of these controller designs. Integrating this C++ application with Matlab would combine the easy design functionality of the Matlab "Robust Control Toolbox" with a high performance Real Time system. The Matlab environment can be easily integrated into a C++ application using the Matlab ActiveX server. A more generic design of the Real Time software will also be required. If the PWM algorithmis altered to be of the form:

$$PWM[i]_j = \begin{bmatrix} Kp_{i,1,1} & Kp_{i,1,2} & \cdots & Kp_{i,1,n-1} & Kp_{i,1,n} \\ Kp_{i,2,1} & Kp_{i,2,2} & & Kp_{i,2,n-1} & Kp_{i,2,n} \\ \vdots & & \ddots & & \vdots \\ Kp_{i,n-1,1} & Kp_{i,n-1,2} & & Kp_{i,n-1,n-1} & Kp_{i,n-1,n} \\ Kp_{i,n,1} & Kp_{i,n,2} & \cdots & Kp_{i,n,n-1} & Kp_{i,n,n} \end{bmatrix} \begin{bmatrix} e_{i,j} & e_{i,j-1} & \cdots & e_{i,j-n+1} & e_{i,j-n} \end{bmatrix} \quad 0 \le i \le 5$$

where j is the current iteration,

n is an arbitrary number of previous arguments that will be stored

e is the error between the actual position and the desired position for each iteration.

and the K matrix is a shared memory object, then entry of discrete time equations at runtime would be possible. The system could be made even more generic by allowing the number of channels i to be set at runtime (of course the number of channels will be limited by the hardware). A discrete time controller could be designed in the Matlab environment, and then loaded directly into the Real Time controller.

7.4 Educational Use of Robot

The current set up has uses for educational purposes. Frequency response data can be read from the system, and a PI controller for the system can be tuned. This is useful for teaching classical control techniques, specifically using loop shaping to create parameters for a PI controller. Depending on the level of students the Real Time

section of the code could be easily altered, giving the students some experience with programming in C/C++.

7.5 Comparison with DSP

This project could have been done without the use of a RTOS, instead a dedicated DSP could have been used. An example of such a solution would be a Heron DSP processor and FPGA module mounted on a PCI carrier board. This is a standard set-up that is available from Traquair Data Systems. The use of a DSP would provide improved system performance (higher sampling rates and reduced latencies). However the downside of using DSP's is that the IPC is more complicated (i.e. communication between the real time controller and the GUI), and programming a DSP also requires greater knowledge of computer hardware and specific embedded programming skills. Development of the real time software on a RTOS offers the advantage of using familiar win32 programming, and an abstract treatment of the hardware. The use of a DSP based system would be a preferable option for systems where an extremely high sampling rate is required, or when a volume production of the system is required. The use of a RTOS such as RTX has its uses in developing one off, test or experimental real time systems, or when a very flexible system set-up is required.

8 Conclusion

An advanced flexible control system has been retrofitted to a Unimate 2000B Robot using a cheap PC, a simple circuit board and an FPGA. Software has been developed that will allow the robot to track paths, a significant improvement over the point-to-point control system that was originally fitted to the robot. A PI algorithm has been used for the controller, with each axis treated as a SISO system. This control scheme has proved to be adequate for path control.

Previous work on this project [Yang *et al.*2000] using Real Time Linux as the operating system and recorded interrupt latencies of 70μs on a 486-66MHz PC. In this application interrupt latencies were cut to 28μs, and the sampling rate was increased from 1kHz to 6kHz (although a faster computer is being used in this application - 233MHz Pentium II). Hardware difficulties encountered in previous work were also corrected and path planning and evaluation added.

Using Windows NT with RTX as a real time operating system as a platform for running real time procedures offers a lot of advantages, including:

- The use of Integrated Software Development tools such as Microsoft Visual Studio,
- control system development tools such as Matlab,
- and the ability to interface with other windows compatible applications such as SolidWorks.

While using a RTOS on a general purpose PC might not offer the same performance as a DSP, in applications like this where a sampling rate of 6kHz is satisfactory, the added difficulties in developing embedded software would outweigh the gains in system performance (sampling frequency and interrupt latencies).

The generation of a model based adaptive controller, and optimisation of the trajectory generation are two areas where improvements could be made on the current system. If the Robot is to be used for educational purposes it may also be desirable to provide closer integration with the Robust Control Toolbox in Matlab.

The research into robotics and real-time operating systems reported in this thesis has shown that a comprehensive system capable of meeting the current and future needs of robotics research has been developed. Further work can use the system developed to explore other path planning strategies and software development for teaching and research.

9 References

[Craig, 1989] Introduction to Robotics: Mechanics and Control, John J. Craig, ISBN 0201095289 Addison-Wesley.

[Dunlop et. al 2000] Dunlop G R, Cree A G, Murphy J D J and Phillips J P, Programmable gate arrays for hard real-time control. Proc. Mechatronics and Machine Vision in Practice, M2VIP, Ed. Billingsley J, Sept. 19-21, Hervy Bay, Australia, Research Studies Press Ltd, ISBN 0 86380 261 3 pp 343 - 348.

[Dunlop et. al 1999] Dunlop G R, Cree A G, Katzir S and Yang J L C, Machine control with real time Linux, Proc. Control 99, Auckland, NZ, (32) pp1-6.

[Gillies, 2001] Dedicated Systems FAQ http://www.dedicated-systems.com/encyc/

[James, 1992] Glyn James,Modern Engineering Mathematics.

[Koopman, 1993] Philip J. Koopman Jr, Perils of the PC Cache, Embedded Systems Programming, 6(5), May 1993, pp. 26-34.

[Kreyszig, 1999] Advanced Engineering Mathematics 8th ed, Erwin Kreyszig. ISBN 0471154962 John Wiley, c1999.

[Sciavicco and Siciliano,1996] Modelling and Control of Robot Manipulators Lorenzo Sciavicco & Bruno Siciliano.

[Yang, 2000] J Yang, ME Thesis, University of Canterbury, Christchurch New Zealand